Attention-based neural re-ranking approach for next city in trip recommendations

Aleksandr Petrov
Edinburgh, United Kingdom
firexel@gmail.com

Yuriy Makarov
Saint-Petersburg, Russia
lvoursl@gmail.com

ABSTRACT

This paper describes an approach to solving the next destination city recommendation problem for a travel reservation system. We propose a two stages approach: a heuristic approach for candidates selection and an attention neural network model for candidates re-ranking. Our method was inspired by listwise learning-to-rank methods and recent developments in natural language processing and the transformer architecture in particular. We used this approach to solve the Booking.com recommendations challenge [3]. Our team achieved 5th place on the challenge using this method, with 0.555 accuracy@4 value on the closed part of the dataset.

CCS CONCEPTS

• **Information systems** → Recommender systems; • **Computing methodologies** → Neural networks.

KEYWORDS

recommender systems, neural networks, learning-to-rank, attention neural networks

1 INTRODUCTION

1.1 Problem description

Next city in a trip recommendations is an important applied problem. When a user plans their trip, the ability to correctly predict next destination can directly benefit the user, saving them time on planning. The benefit of the user is important for the booking service provider - if the user gets precise recommendation, they are more likely to book it using the same platform and therefore increase the company revenue. In late 2020 Booking.com, the largest hotel reservations service provider, released a dataset and launched a public competition [3] with the goal of achieving the best Accuracy@4 metric. We built an attention based model for this competition and achieved 5th place with final score of 0.555 Accuracy@4 on the closed test set.

1.2 Dataset and target metric

The dataset, released by booking.com, consists of two parts: *train* and *test*. Both parts include anonymized hotel checkins with following features: userId, checkinDate, checkoutDate, cityId, deviceClass, affiliateId, bookerCountry, hotelCountry, utripId. CityId and hotelCountry of the last checkin in the trip were masked in the *test* part of the dataset and were used for scoring by the competition organizers.
The goal of the competition was to generate recommendations list of 4 cities. The organizers used Accuracy@4 as the main contest metric, which essentially is a percentage of times when our recommendations contained correct cityId.

2 ATTENTION-BASED RERANKING MODEL

Accuracy@4 is a ranking metric; therefore, we solved the problem as a *ranking* problem. Accuracy@4 is not a very stable metric, as it does not take into account the order of the elements in the recommendations list and takes into account only 4 scores for each recommendation. Instead, we chose another popular ranking metric - NDCG@40[4] as our main optimization target. This metric is more stable as it considers the order of the elements in the list and takes into account 40 scores rather than 4.

We chose the 2-stages approach, which is popular for solving ranking problems. In the first stage, we selected 500 candidates from all cities using a mixture of simple models. In the second stage, we re-ranked the candidates using a self-attention neural network. We used LambdaRANK[2] approach for the optimization, as it optimizes the NDCG metric directly.

Our main model was inspired by the transformer architecture [9]. The architecture is designed for language processing and achieved state-of-the-art results in many text analysis tasks. The idea of utilizing a language model comes from the fact that a sequence of cities in the trip has a very similar structure as a sequence of words in the text. This idea was already successfully applied to recommender systems; see [8] for example.

On the high level, the model works as follows:

(1) Select candidates for ranking.
(2) Generate candidates matrix C. Each row of this matrix contains a vector representation of the corresponding candidate. The vector consists of learnable embeddings as well as engineered features.
(3) Generate vector representations of all known cities in the trip T using transformer-like architecture
(4) Generate vector representation of the last city in the trip F. Even though the target city itself is unknown, the dataset contains such features as checkinDate, checkoutDate, deviceClass, affiliateId, bookerCountry, which we can use for the prediction.
(5) Generate scores for the candidates. To do this we use MultiHeadAttention mechanism [9] between Candidate Cities and Trip and then calculate the final score using dot product with the encoded features vector:

$$Scores = MultiHeadAttention(C, T) \cdot F \quad (1)$$

2.1 Labels generation

The model training process requires labeled data. To get the labels for each trip, we performed the following procedure:

(1) Sort actions in the trip by the time of check-in.
(2) Choose a random fraction of the last cities in the trip as target cities. We re-split the data on each training epoch. This

technique allowed us to generate multiple training samples out of a single trip.

(3) Assign label score 2^{-i} for i^{th} city in the target fraction. The idea is that each city from the target fraction is relevant as a trip continuation, but we want our model to rank higher the cities that the user visited in the near term. The intuition behind giving exponentially diminishing weights for future trips is based on the fact that the cities that the user will visit in the short-term future are more likely to be related to the cities the user visited recently.

2.2 Candidates selection

Our model uses the LamdaRANK ranking optimization approach[2], which involves heavy computations and only can generate scores for a limited amount of candidate cities at once. Given that, we needed to generate good candidates using some simpler approaches. To train the model reasonably fast, we limited the number of candidates to 500 (out of 39870). We used a mixture of simpler models to fill the set of candidates.

Basic models for candidates selection. This paragraph describes our basic models and heuristics that we used to produce a list of the candidate cities. During each training epoch, we randomly chose 10000 trips to train our main neural model to prevent overfitting, and we used the rest of the training set to re-train baseline models and generate features. Here is the list of the models that we used to generate candidates:

(1) *All cities from the trip.* We have found that quite frequently, the user already visited the last city from the trip previously in the same trip, so all the cities from the trip are good candidates. We added all the cities from the trip to the set of candidates.

(2) *TransitionChain* – this model utilizes a sequential nature of the data.
Let's assume that the total number of cities in the dataset is equal to M, and each trip has K cities, where K-th city is a target city. We can create transition matrix $T \in \mathcal{R}^{M \times M}$, which we can fill in a way, described in Algorithm 1.
The prediction process is straightforward: we take all cities in the trip (except target) and sum up all lines in the matrix corresponding to these cities, then we took cities with the highest scores and used them as predictions.
We generated recommendations for the trip using TransitionsChain and added the resulting recommendations into the candidates set until it reached size 150.

(3) *BookerTripCountryTop* – This model generates the most popular cities in trips from the users from a particular country who had the same country of the last city in the trip. We added recommendations from this model to the candidates set until we got 350 candidate cities.

(4) *LastCityCountryTop* – This model calculates the most popular next city based on the country of the last city in the trip and generates them as recommendations. We added recommendations from this model to the candidates set until we got 500 candidate cities.

Algorithm 1 Transition Matrix (T) Filling

```
1:  procedure fillTransitionMatrix
2:      $T \leftarrow$ zero matrix with size $M \times M$
3:      for $current\_trip$ in $trips$ do:
4:          $last\_city \leftarrow current\_trip[-1]$
5:          $prev\_cities \leftarrow current\_trip[:-1]$
6:          for $city$ in $prev\_cities$ do:
7:              $T[city][last\_city]$ += 1.
8:          end for
9:      end for
10: end procedure
```

In some cases, we were not able to fill candidates set using the models above. For example, when a user visited very unpopular cities, we don't have enough statistics in the training set. In this case, we added candidates to the candidates set from two additional models:

(5) *BookerCountryTop* – This model calculates most popular cities among users from a specific country. We added candidates from this model until we have 500 candidates in the set.

(6) *GlobalTop* – This model generates the most popular cities in the training set as recommendations. We added candidates from this model if all previous models could not fill the set of candidates.

Our experiments have shown that this heuristic includes the right candidate into the set of candidates with 90% probability.

2.3 Generate candidate's features

For each candidate, we generated a vector representation that included the following features:

- City's popularity globally and especially for booker's country
- City's popularity according to current month and year
- City's score from TransitionsChain recommender.
- Binary flag: is the candidate same city as the first city in the trip
- Binary flags: for each of the last 5 cites in the trip - is the candidate the same as this city from the trip
- Candidate's cosine similarity with last 5 cities in a trip based on their co-occurrence in train data.

2.4 Trip Encoding

Our model used transformer[9]-like architecture to encode cities from the trip history. We represented each city in the trip as a vector, that included following parts:

(1) City embedding - 32 dimensions, learnable.
(2) Booker country embedding - 32 dimensions, learnable.
(3) Hotel country embedding - 32 dimensions, learnable (share weights with the booker country embedding)
(4) Affiliate id embedding - 5 dimensions, learnable.
(5) City-in-trip features. Manually engineered features, that include:
   - Number of nights between check-in and check-out dates
   - Is this trip over the weekend or not?

- Is the current city in the same country as previous ones?
- City's check-in and check-out day of week and day of the year
- Is booker's country equal to hotel's country?
- Check-in year (3 features, one-hot encoded)
- Checkin month (12 features, one-hot encoded)

Overall, each city in the trip was represented using 115 parameters. We stacked the cities' representations in the trip and padded the matrix to the shape (50, 115), where 50 is the maximum possible number of cities in the trip.

To get a better semantic representation of the cities in the trip, we encoded each city using a single dense layer with 115 dimensions.

Similarly to the original transformer architecture, we combined city representation with positional embeddings. In the original paper[9] the authors use fixed embeddings, based on *sin* function. We replaced this with two learnable embeddings, representing the city's position from the beginning and the end of the trip. These embeddings are then concatenated, passed through a dense layer, and multiplied with city representation vectors.

To model interactions of the cities in the trip, we added three transformer-like blocks. The original transformer block consists of multi-head attention, feed-forward layer, residual connection, and layer normalization. Our transformer-like block has a similar architecture with only a minor difference: we used Multiplication for residual connection instead of Sum. Our experiments have shown that the network with Multiplication instead of Sum required fewer epochs to train. Figure 3 shows architecture of the Transformer-like block used in our architecture.

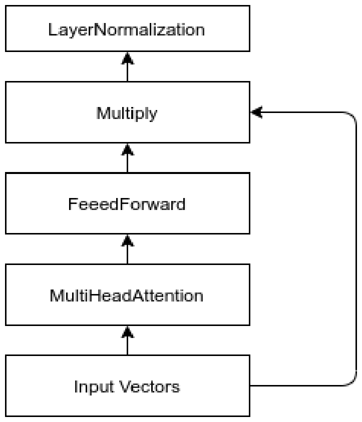

Figure 1: Transformer-like block

We then used the Transformer-like part of the model to model interactions between candidate cities and cities from the trip. In addition to the features matrix described in section 2.3, we added city embedding and country embedding to each candidate. These embeddings share weights with the city embeddings and country embeddings from the trip-encoding part of the model. The candidates were additionally encoded using a feed-forward layer. We also added one transformer-like block to model the relationship between candidates. The idea is that the candidates are not independent, and the model should score candidates knowing about other candidates. A similar idea of modeling interactions between candidates for recommendations is described in [7]

We used one multi-head attention layer to model interaction between encoded user trips and encoded candidate cities. We hypothesize that this output layer contained semantically rich representations of the (trip, candidate) pairs.

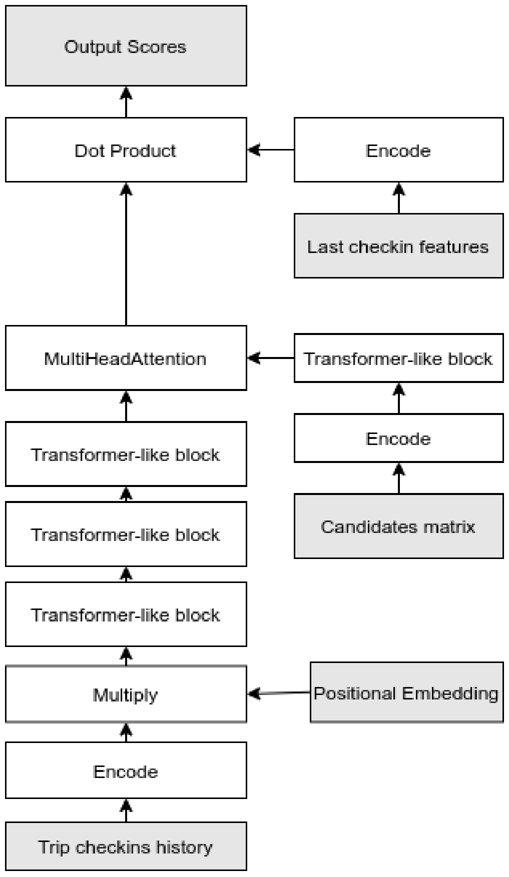

Figure 2: High-level model architecture

To generate the model's final scores, we used dot product operation with encoded features of the target city. The dataset contains all parameters of the target city, except cityId and countryId. We encoded these parameters using the same manually engineered features as described in section 2.3 and encoded them using a dense layer.

2.5 Model training

Our model's goal was to rank candidates according to the label score, which constitutes a ranking task. The problem of direct optimization of the ranking metrics, such as Accuracy@4 or NDCG, is difficult because these metrics only depend on the order of the items and therefore are not smooth and can not be directly optimized gradient-descent technologies. To overcome this problem, we used the LambdaRANK approach described in [2]. The method's idea is that instead of calculating the loss function and calculating its gradients, one can directly calculate gradients. The authors of [2] propose generating such gradients for optimizing the NDCG metric. We implemented this method and used it as our main optimization approach.

We optimised our network using Adam[5] optimiser. We trained our model using 50-epochs early stop criteria on val_accuracy_at_4 metric.

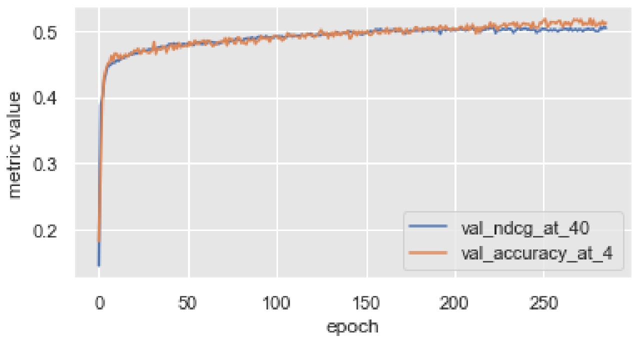

Figure 3: Validation performance during training process

Figure 3 illustrates model performance on the validation set during the training process. As one can see from the plot, the NDCG@40 (metric optimized by the LambdaRANK approach) and Accuracy@4 (contest target metric) demonstrate high correlation, suggesting that optimizing one metric leads to optimization of the other.

We used Keras and TensorFlow[1] libraries to train our neural networks. Training of the model took 7 hours on GeForce GTX3090 GPU.

3 EVALUATION AND RESULTS

In this section, we describe the validation scheme, metrics and analyze our results.

3.1 Validation scheme

Our validation scheme was based on trips. We used both *train* and *test* datasets provided by the competition organizers to train our model. We randomly allocated 4000 trips to the hold-out set, 4000 trips to the validation set, and the rest to the training set. We tried to predict the last city in the trip based on all previous cities' information and available information about the last city in the validation and hold-out sets.

3.2 Metrics

We used the following metrics to evaluate our models:

- `Accuracy@4`: competition organizers proposed this metric. The idea of Accuracy@4 is quite simple: it is equal to 1 if the target city in our top-4 prediction and 0 otherwise.
- `NDCG@40`: according to our experiments, NDCG@40 is a more stable metric than previous ones. We also found that NDCG@40 and Accuracy@4 have a quite strong correlation (Figure 3.). One can find the details about the NDCG metric in [4]
- `Leaderboard`: this metric is essentially the Accuracy@4 metric, calculated by the organizers on the closed part of the dataset. We could only calculate this metric for two models by the competition's conditions - the one that we sent for the intermediate leaderboard and the final submission.

Table 1: Models comparison

Model	Accuracy@4	NDCG@40	Leaderboard
GlobalTop	0.058	0.091	-
TruncatedSVD	0.261	0.261	-
LastCityCountryTop	0.372	0.358	-
TransitionChain	0.440	0.429	-
SelfAttention	0.509	0.491	0.514
LambdaMART	0.514	0.485	-
Reranking Attention	**0.542**	**0.513**	**0.555**

3.3 Model performance against baselines

We used the following baselines to evaluate quality of our model:

- *GlobalTop, LastCityCountryTop, TransitionChain* - baselines used as part of our candidates selection process (see section 2.2)
- *TruncatedSVD* - a popular method of solving recommendations problem [6].
- *SelfAttention* - end-to-end version of our model. The model's architecture consists of a transformer-like part of our model, followed by three dense layers, the last of which generates recommendation scores for all cities in the dataset. The main difference compared with the reranking attention model is that this model doesn't use manually engineered candidate features. This makes the model simpler, and therefore the model can produce scores for all cities in the dataset rather than a small number of candidates.
- *LambdaMART* - Gradient boosting trees implementation of the LambdaRANK approach. We used the same candidate generation procedure and same manually engineered features described in section 2.3. We used LambdaMART implementation from LightGBM lilbrary[1].

Table 1 contains results of the comparison. Our final model demonstrates statistically significant improvement over baselines on the Accuracy@4 metric (p-value < 0.01 in two-tailed Z-test).

4 CONCLUSION

As this challenge has shown, combining the listwise ranking methods and neural models is a viable idea that allowed us to achieve a high score on the contest. The same optimization approach can be used with other architectures and solve many other ranking problems in the information retrieval area, including recommender systems and search results ranking.

[1]https://lightgbm.readthedocs.io/en/latest/

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.

[3] Dmitri Goldenberg, Kostia Kofman, Pavel Levin, Sarai Mizrachi, Maayan Kafry, and Guy Nadav. 2021. Booking.com WSDM WebTour 2021 Challenge. https://www.bookingchallenge.com/. In *ACM WSDM Workshop on Web Tourism (WSDM WebTour'21)*.

[4] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.

[5] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[6] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[7] Changhua Pei, Yi Zhang, Yongfeng Zhang, Fei Sun, Xiao Lin, Hanxiao Sun, Jian Wu, Peng Jiang, Junfeng Ge, Wenwu Ou, et al. 2019. Personalized re-ranking for recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 3–11.

[8] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (Beijing, China) *(CIKM '19)*. ACM, New York, NY, USA, 1441–1450. https://doi.org/10.1145/3357384.3357895

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762* (2017).