Location Embeddings for Next Trip Recommendation

Amine Dadoun
EURECOM, Sophia Antipolis, France
Amadeus SAS, Biot, France
amine.dadoun@eurecom.fr

Raphaël Troncy
EURECOM, Sophia Antipolis, France
raphael.troncy@eurecom.fr

Olivier Ratier
Amadeus SAS, Biot, France
olivier.ratier@amadeus.com

Riccardo Petitti
Amadeus SAS, Biot, France
riccardo.petitti@amadeus.com

ABSTRACT

The amount of information available in social media and specialized blogs has become useful for a user to plan a trip. However, the user is quickly overwhelmed by the list of possibilities offered to him, making his search complex and time-consuming. Recommender systems aim to provide personalized suggestions to users by leveraging different type of information, thus assisting them in their decision-making process. Recently, the use of neural networks and knowledge graphs have proven to be efficient for items recommendation. In our work, we propose an approach that leverages contextual, collaborative and content information in order to recommend personalized destinations to travelers. We compare our approach with a set of state of the art collaborative filtering methods and deep learning based recommender systems.

CCS CONCEPTS

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

KEYWORDS

Recommender Systems, Neural Networks, Knowledge Graph, Embeddings, Tourism

ACM Reference Format:
Amine Dadoun, Raphaël Troncy, Olivier Ratier, and Riccardo Petitti. 2019. Location Embeddings for Next Trip Recommendation. In *Companion Proceedings of the 2019 World Wide Web Conference (WWW '19 Companion), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3308560.3316535

1 INTRODUCTION

Traveling is no longer considered to be just a need and it has clearly become a desire. Users became exposed to many inspirational tourism posts and advertisements in social media, travel forums, travel agencies and airline websites. Although inspirational, many of these posts might not fit a particular user's profile and, thus, they may not be relevant to him. In recent years, destination recommender systems (DRSs) have been proposed to suggest a ranked list of destinations, sometimes composed of sights, events and cities to visit, based on information provided by the user [10, 13, 27]. Recommender systems can also take contextual information into account, for example, by leveraging event-based social networks data [15]. Location-based Social Networks (LBSNs) allow users to publicly or privately share their position by performing a check-in when visiting a certain venue or Point-of-Interest (POI). Leveraging this data enables to first know what a city is best characterized by (restaurants, sport events, museums, parks, etc.), and then to identify the user's interests [17]. Finally, a user profile can be enriched with his booking history.

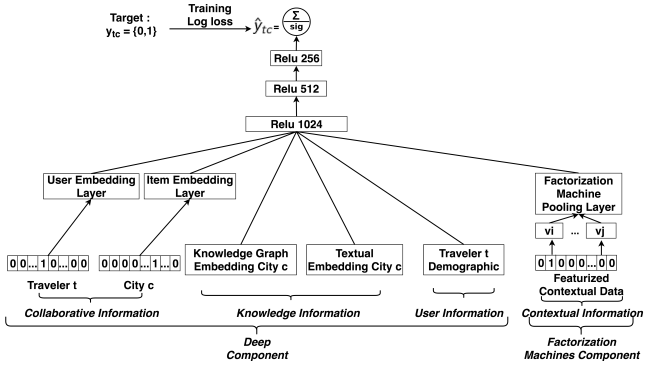

Figure 1: Deep Knowledge Factorization Machines architecture for next trip destination.

Recent works have illustrated the effectiveness of using knowledge graph embeddings (KGE) [19, 24, 28] and neural networks [2, 7, 8] for item recommendation. In this paper, we propose a Deep Knowledge Factorization Machines (DKFM) architecture to recommend destinations. Our approach relies on learning *i)* a representation of cities using different data sources including Wikipedia and LBSN, *ii)* the long-term user's behavior using his booking history and *iii)* a representation of the context associated with each past trips. More specifically, we combine textual embeddings representing cities based on their wikipedia content description with knowledge graph embeddings that represent cities' characteristics in terms of venue check-ins made by users in LBSNs. We also combine two existing deep learning based recommender systems [2, 7] to build our so-called DKFM architecture (Figure 1) which takes as input content, collaborative and contextual information related to user bookings.

The main contributions of our work are:

WWW '19 Companion, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6675-5/19/05.
https://doi.org/10.1145/3308560.3316535

(i) We leverage three types of information (collaborative, content and contextual) in order to recommend a next trip.
(ii) We present a deep neural network model that takes as input these three different types of information.
(iii) We perform an empirical comparison on a real-world dataset of our DKFM architecture with state-of-the-art collaborative filtering methods and deep learning based recommender systems.

The rest of the paper is organized as follows. Section 2 provides a literature review of the related work. Section 3 introduces some preliminaries concepts, the recommendation problem and the dataset we use for the experiments. In Section 4, we present the approach to build the DKFM architecture. Section 5 presents the experiments carried out to show the effectiveness of our model. Finally, in Section 6, we provide some conclusions and we discuss some future work.

2 RELATED WORK

This section provides a literature review of recommender system in the tourism domain, which has known growing interest from many academic and industrial researchers, trying to tackle several recommendation tasks such as recommending POIs, suggesting sequence of attractions/activities to do or proposing a complete trip plan. This section presents state-of-the-art methods that utilize knowledge graph embeddings and deep learning methods for item recommendation.

Recommender System in Tourism: Tourism includes traveling for business or leisure. It involves complex decision-making from travelers to select destinations, hotels, events, activities, etc. On the other side, travel industry players (e.g. travel agents) are helping travelers to find the most suitable options. Early works have focused on personalized techniques in order to provide recommendation based on user's preferences and interests [22]. More specifically, the idea is to match items in a catalogue of destinations with the user needs, and interests expressed by the offered language. Kiseleva et al. [13] proposed a multi-criteria rating system (MCRS) based on naive bayes approach in order to recommend travel destinations in a hotel booking platform[1]. MCRS are based on which aspects a user liked for a given item, while classical recommendation systems are based on a single rating (e.g. giving a rating for a movie). Wolfgang et al. [26] proposed an approach to generate sequence of POIs when visiting a city based on three user's inputs: start and end point plus interests. However, user preferences or item characteristics are in many cases insufficient to have accurate recommendation. Macedo et al. [15] have proposed to use contextual signals provided by LBSNs such as time or location for events recommendation. In our work, we combine three different types of input: the traveler-destination interaction which is represented by a booking, the context of each booking, and the content that characterizes each destination, to suggest destinations to travelers.

Knowledge Graph Embeddings for Items Recommendation: A Knowledge graph embedding is a representation of a knowledge graph's component into continuous vector space. The idea is to ease the manipulation of graph components (entities, relations) for prediction tasks such as entity classification, link prediction or recommender systems [28]. A survey of approaches and applications for knowledge graph embeddings was done by Wang et al. [25]. Two main approaches exist in order to learn knowledge graph embeddings from a KG: translational distance models where the goal is to minimize the distance between neighbors entities in the graph; semantic matching models which are based on the semantics of the graph components compute a similarity score that measures the semantic similarity between each entity in the graph. In order to enrich the collaborative information represented by the user-item interaction with additional information, knowledge graphs have been used to provide knowledge about items and/or users to enhance the performance of the recommendation. In [28], the authors used a knowledge base containing different external resources: textual information, visual information and structural information (knowledge graph embedding) to enrich the user implicit feedback for items recommendation. In [18], the authors used the concept of property-specific knowledge graph to learn embeddings based on node2vec [5] of each subgraph and then used a ranking function to provide items recommendation. In our work, we use a knowledge graph from LBSN's check-ins in order to learn knowledge graph embeddings that represent cities.

Deep learning based Recommender System: In the recent years, deep learning has demonstrated its effectiveness when applied to information retrieval and recommender system. In [4], the authors used a multilayer perceptron that takes as input the (user, item) interaction and learn user and item embeddings. In [8], the authors combined a multilayer perceptron with a generalization of matrix factorization represented by a single layer perceptron. In [2], the authors proposed wide and deep learning model for app recommendation[2]. The wide learning component is a single layer perceptron which enables to capture memorization and the deep learning component is a multilayer perceptron which enables to capture generalization. In [6], the authors combined factorization machines and multilayer perceptron. The idea is to model the high-order feature interactions via multilayer perceptron and low-order interactions with Factorization Machine [20]. In [7], the authors proposed to use a pooling layer that computes the first order feature interaction term in factorization machines formula [20], then feed the obtained vectors in a multilayer perceptron. Other neural network architectures have been used for recommendation, such as Recurrent Neural Networks (RNNs) (e.g. session-based recommendation) or Convolutional Neural Networks (CNNs) used for example to capture images representation in order to enrich item representation.

Our work focuses on feed-forward neural networks. We combine two existing deep learning based recommender systems [2, 8] to build our recommender system.

3 DATA & PROBLEM FORMULATION

In this section, we first provide definitions of some concepts useful to build our recommendation system. Then, we present the next trip recommendation task. Finally, we present the dataset used to address the next trip recommendation.

[1]Booking.com: https://www.booking.com

[2]Google Play Store: https://play.google.com/store

3.1 Preliminaries

We define a Knowledge Graph similarly to what has been done in [18].

Definition 1. A knowledge graph is a set $K = (E, R, O)$, where E is the set of entities, $R \subset E \times \Gamma \times E$ is a set of typed relations between entities and O is an ontology.

In recommender system realm, there are two different types of feedback: explicit feedback where the user gives a rating on how he liked the item or not, and the implicit feedback where we know only the interest of a user for an item. Concretely, in our case, the implicit feedback denotes the fact that a traveler t visited a destination d.

Definition 2. Given a matrix $\mathbf{T} \in \mathbb{R}^{n \times m}$, where t_{ij} is the number of times the traveler i traveled to destination j, n the number of travelers and m the number of different destinations. We define the traveler binary feedback matrix $\mathbf{R} \in \mathbb{R}^{n \times m}$ as follows:

$$r_{ij} = \begin{cases} 1 & \text{if } t_{ij} > 0 \\ 0 & \text{otherwise .} \end{cases} \quad (1)$$

3.2 Problem Formulation

Definition 3. Given a traveler, his demographics information (age, nationality, etc.), his historical bookings and the contextual data related to those bookings (day of week, number of passengers, stay duration, etc.), we aim to recommend to this traveler a ranked list of destinations he would like to go to. A destination is represented by a city that has an airport. The Figure 2 illustrates the recommendation task we want to tackle.

Figure 2: The recommendation task is to predict a ranked list of next destinations for a traveler given his historical bookings.

3.3 Data Preprocessing

Experiments were conducted on a real-world dataset of bookings from the so-called CEM[3] database, an Amadeus database containing bookings over a dozen of airlines. Each booking is stored using Personal Name Records (PNR) information. The PNR is created at reservation time by the airline and contains information about the purchased ticket (travel itinerary, traveler demographics information, payment information, ancillary services bought with the air ticket, etc.). The considered dataset contains 4.8 Million bookings for 814.919 unique travelers[4].

[3]CEM: Customer Experience Management

[4]Statistics of the pre-processed dataset are given in Tables 3 & 4

Table 1: Features used for business/leisure classification

Feature Name	Type	Range
Number Passenger	Numerical	{1..9}
Stay Duration	Numerical	[0,99]
Saturday Stay	Binary	{0,1}
Purchase Anticipation	Numerical	{0..364}
Age	Numerical	{0..99}
Gender	Categorical	{Female, Male, Unknown}

Table 2: Business/Leisure classification performance

Metric	Score
Accuracy	0.87
Precision	0.87
Recall	0.91

Customer segmentation model: The approach to recommend destinations to business/leisure travelers is expected to be different as explained in [3]. In our work, we focus only on recommending travels for leisure purpose. Hence, we need to split the bookings into business/leisure segments. Given an historical dataset of bookings that are already labeled into business/leisure travels, we build a Random Forest based classifier in order to classify our bookings. Table 1 shows the dataset's features used for this classification task. In order to train our classifier, we use an existing dataset that contains bookings already labeled (Business/Leisure). This dataset was collected from travel agencies from February 2014 to February 2017 and contains 122,242 bookings (60% leisure). The classifier (Random Forest) was tuned using grid-search algorithm over the following hyper-parameters: maximum tree depth taken in {5, 8, 10}, maximum used features taken in {0.6, 0.65, 0.7, 0.75}, minimum samples in leaf taken in {1, 2}, number of trees taken in {100, 150, 200}. Finally, to evaluate our classifier, we used a 10 Fold Cross Validation method by splitting our dataset into training and test sets (90 % training, 10 % test set) and compute the accuracy, precision and recall metrics for the tuned classifier. We also compute the importance of each feature for the classification based on the relative information gain of each feature; The number of passenger is the most important feature for this classification task.

The classifier was then used to classify bookings of the considered dataset into Business/Leisure bookings. We keep only leisure bookings for our work. We obtained 2 Million bookings made for 629.156 unique travelers, which represent 42% of the whole dataset.

Data Filtering for Recommendation: Despite the huge amount of available bookings that could be used to build the recommendation system, we have a very sparse traveler feedback matrix. The sparsity of the traveler binary feedback matrix $\mathbf{R} \in \mathbb{R}^{n \times m}$ is defined as follows:

$$\rho(\mathbf{R}) = 1 - \frac{\#interactions}{m \times n} \quad (2)$$

where, n is the number of users and m is the number of different destinations.

Table 3: Statistics of the preprocessed dataset

#Feedbacks	#Interactions	#Cities	#Travelers	Sparsity
304019	152547	119	26019	95%

Table 4: Statistics of the preprocessed dataset

Variable	Min	Max	Std	Mean	Median
#Visiting same city	1	354	3.34	2	1
#Travelers per city	20	19496	2452	1282	293
#Cities per traveler	5	37	1.49	5.86	5

In fact, for this dataset, the sparsity is 99.6%. Moreover, more than 65% of the travelers have traveled only two times. Similarly to [8, 9], in order to cope with this issue, we keep only travelers that have at least **5** different destinations in their history, and destinations that were visited at least **20** times. After applying this last filter, we obtained a dataset containing 304.019 booking for 26.019 unique travelers. It is important to note that an interaction represents the fact that a traveller has been to a destination at least once, which is different from feedback, which represents the number of times a traveller has been to that destination. Tables 3 & 4 represent statistics of the pre-processed dataset.

4 APPROACH

Our approach is to leverage data from different sources to improve the recommendation by enriching implicit interaction between traveler and destination with external knowledge. Moreover, the context in travel is an important factor to consider when doing recommendation, thus, we also add contextual information related to a given booking. We combine a deep component which is a multilayer perceptron that takes as input the implicit interaction and the content information, with a factorization machines component that takes as input contextual data. The deep and factorization machines components were combined by concatenating:

- Traveler and city embeddings (the destination is represented by a city);
- Textual and knowledge graph city embeddings;
- User demographics information;
- Contextual feature vectors computed by the pooling operation (see Section 4.4).

The concatenated vectors are fed into a multilayer perceptron. The two components are jointly trained using backpropagation algorithm to learn the weights of the deep and factorization machines components, and also the traveler and city embeddings. In this section, we first present how the destinations are enriched with external knowledge resources, and present the different components of our model. Finally, we present how we combine the two existing deep learning based recommender systems to build our model.

4.1 Textual Embedding

In this subsection, the goal is to explore the different ways to build a textual representation of the cities based on the content of their Wikipedia pages. The first step is to retrieve all the Wikipedia pages of the 119 cities covered in our dataset. To do so, we used the Wikipedia Python API[5]. Once all documents describing these cities have been retrieved, we need to define a method to construct an embedding of each document. In the recent years, many competing algorithms were proposed to learn sentence or document representation. In [12], the authors proposed to learn unsupervised sentence embeddings based on RNN encoder-decoder which is trained to reconstruct the surrounding sentences from the current sentence similarly to what is done in skip-gram model for word embeddings. In [14], the authors proposed a faster way to learn unsupervised sentence representations by reformulating the problem as a classification task, where the classifier has to choose the right next sentence among a set of possibilities. While these approaches have shown good performance, simple baseline like averaging pre-trained word embeddings give also strong results. We propose to encode a sentence in a weighted sum of word vectors, where the weight of each word vector corresponds to the term frequency-inverse document frequency (TF-IDF) of the word. We used the fastText pre-trained word vectors [16]. The word vectors were trained using Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset.

4.2 Knowledge Graph Embedding

Similarly to the previous subsection, we explore the different methods to construct a city representation based on knowledge graph. First, we need to build a knowledge graph that contains characteristics of cities. In [17], the authors used LBSNs to build users' trails. A trail is a succession of check-ins made by a user in venues. Each venue is categorized using Schema.org[6] (Restaurant, Civic structure, etc.) and the Foursquare category which is more detailed (Italian restaurant, Indian restaurant, etc.). The authors released the Semantic Trails Datasets[7] which contain two datasets. The first dataset contains check-ins collected in the temporal interval going from 03-04-2012 to 16-09-2013, while the second one contains check-ins collected going from 03-10-2017 to 19-10-2018. To build the Semantic Trail Knowledge Graph in Figure 3, we used both datasets. The knowledge graph represents the interaction user-venue, through the property 'visiting' as well as the relations between the venue and the other entities, namely: category, schema and city.

In [19], the authors presented an empirical comparison of translational distance models for items recommendation. The results have shown that TransE [1], the model with the least parameters in comparison with other translational distance models [25] obtained the best scores over a set of metrics. We propose to use TransE to learn embeddings for the entities and relations in the knowledge graph, and extract the cities embeddings. The idea of TransE algorithm lies in minimizing a distance D, between $h + l$ and t, so that $D(h + l, t) \approx 0$, where the triple (h, l, t) corresponds to (head, relation, tail) entities. Finally, in order to match the cities of our dataset with the cities embeddings obtained by the semantic trails knowledge graph, we used their Wikidata id[8].

[5]Python Wikipedia API: https://pypi.org/project/wikipedia/
[6]Schema.org: https://schema.org
[7]Semantic Trails Datasets: https://figshare.com/articles/Semantic_Trails_Datasets/7429076
[8]Wikidata: https://www.wikidata.org/wiki/Wikidata:Main_Page

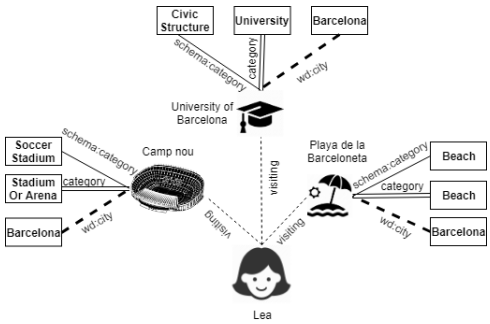

Figure 3: Semantic Trails Knowledge Graph

Table 5: Semantic Trails Datasets

Feature Name	Type	Range or #different values
Trail ID	Numerical	{1..Number of Trails}
User ID	Numerical	{1..Number of Users}
Venue ID	Categorical	~ 4.4 Million
Venue Category	Categorical	934
Venue Schema	Categorical	162
Venue City	Categorical	43833
Venue Country	Categorical	207
Time Stamp	Date	2012-04-03 To 2018-10-19

4.3 Deep Component

As shown in Figure 1, the deep component is a feed-forward neural network, that takes as input the one-hot encoded vector of the traveler and the city (t,c) and transform these two vectors into low-dimensional and dense vectors through the embedding layer which is a single layer perceptron, whose weights are initialized randomly. Weights are updated in backpropagation phase. The user demographics information plus the city knowledge graph and textual embeddings are concatenated with the traveler and the city embeddings to form the input of the deep component.

4.4 Factorization machines Component

In [7], the authors modeled the first order feature interaction factorization machines term by using two layers. The first layer takes as input the vector $\mathbf{x}$ corresponding to contextual information, and create an embedding vector of each feature of $\mathbf{x}$. More formally, the first layer computes a vector $\mathbf{v}_i \in \mathbb{R}^k$ for each feature i, where k is the dimension of features vector. In the second layer, the following pooling operation is performed $\mathbf{x}$:

$$f(x) = \sum_{i=1}^{k} \sum_{j=i+1}^{k} x_i \mathbf{v}_i \odot x_j \mathbf{v}_j \tag{3}$$

where, $\odot$ denotes the element-wise product.

As shown in Figure 1, we use the same two layers in order to compute the factorization machines feature vectors interaction term.

4.5 Deep Knowledge Factorization machines

The obtained vectors from the deep component and the factorization machines component are concatenated and fed in a multilayer perceptron that contains different hidden layers. In each hidden layer l, we perform the following computation:

$$a[l] = Relu(\mathbf{W}[l-1]^T \mathbf{a}[l-1] + b[l-1]) \tag{4}$$

where, $Relu(x) = max(0, x)$ is the rectified linear unit function. It is used as the activation function for each layer of the multilayer perceptron. While there are other functions that can be used as activation function (sigmoid or hyperbolic tangent), Relu function is proven to avoid vanishing gradient problem, and showed better results in the experiments. $a[l-1], W[l-1], b[l-1]$ are respectively the activations, weights and bias of the previous layer (layer l-1).

Finally, at the end of the last hidden layer L, we compute the prediction $\hat{y}_{tc}$ by applying a sigmoid function to restrict the value between 0 and 1 which represents the probability to recommend the city c to the traveler t:

$$P(t, c|\mathbf{X}) = \hat{y}_{tc} = \sigma(\mathbf{h}^T \mathbf{a}[L]) \tag{5}$$

where, $\sigma(x) = \frac{1}{1+e^{-x}}$, h is the weight vector of the last neuron, $\mathbf{X}$ is the input vector of the multilayer perceptron.

The objective function of the backpropagation algorithm is to minimize the logistic loss defined as the negative log-likelihood of the observation (1 if traveler t went to city c, 0 otherwise) given the model's predictions:

$$\begin{aligned} Loss(\hat{y}_{tc}, y_{tc}) &= -log(P(\hat{y}_{tc}|\mathbf{W}, b, \mathbf{h})) \\ &= -(\sum_{u=1}^{n} \sum_{i=1}^{m} y_{tc} \times (log(\hat{y}_{tc}) + (1 - y_{tc}) \times log(1 - \hat{y}_{tc})) \end{aligned} \tag{6}$$

where, $P(\hat{y}_{tc}|\mathbf{W}, b, \mathbf{h})$ is the likelihood function of $\hat{y}_{tc}$, n is the number of users, and m the number of items.

5 EXPERIMENTS

In this section, we will try to give answers to the following research questions with empirical experiments:

(i) **RQ1**: What is the contribution of the deep component?
(ii) **RQ2**: What is the contribution for each input used in the deep component: traveler demographics data, city embeddings?
(iii) **RQ3**: What is the contribution of factorization machines component?
(iv) **RQ4**: How our model perform in comparison with baseline models?
(v) **RQ5**: How the performance of our model is affected by the hyper-parameters?

5.1 Experimental Setup

In this subsection, we present the different settings used to conduct the experiments as well as the baseline models used to perform the empirical comparison.

Dataset: We evaluate our model with the dataset obtained in the section 3.2. The characteristics of the dataset are shown in Tables 3 & 4.

Training & Test Sets: The recommendation task consists in predicting the next trip for a given traveler based on his previous trips. Hence, the dataset must be split in such a way that the test set must contain the last trip for each traveler. To do so, we adopt the leave-one-out strategy used in [8]. Formally, for each traveler, his last trip is used in the test set and the remaining trips are kept for the training set. Finally, n_s random cities where the traveler never went to are considered as negative samples. The experiments showed that n_s was performing well for a value of 3.

Evaluation Metrics: The output of the recommender system is a ranked list that contains all the cities where the user never went to in addition to the city in the test set. Hence, a good recommender system will rank this city in the test set at the top-K (we set K=10). To evaluate, our recommender system, we used two metrics defined as follows:

- **HR@K**: Hit Ratio metric measures whether the relevant city is within the Top K in the ranked list returned by the predictive model:

$$HR@K = \frac{1}{n}\sum_{t=1}^{n}\sum_{j=1}^{K} hit(t, c_j) \tag{7}$$

- **MRR@K**: Mean Reciprocal Rank metric is used to measure how well the predictive model ranked the relevant city against the irrelevant ones:

$$MRR@K = \frac{1}{n}\sum_{t=1}^{n}\sum_{j=1}^{K} \frac{1}{rank(t, c_j)} \tag{8}$$

where, $hit(t, c_j$ is equal to 1 if the traveler t visited the relevant city c_j, $rank(t, c_j)$ is the position of the relevant city c_j in the ranked list, n is the number of travelers.

Baseline Models: We compare our model Deep Knowledge Factorization Machines with a set of baseline methods that include collaborative filtering methods, factorization machines model and also two state-of-the-art deep learning based recommender systems. All the baselines are summarized below:

- **MostPop:** Cities are ranked by their popularity. The popularity of a city is calculated by the number of visits by all the travelers. The top-k popular cities are proposed to every traveler.
- **ItemKNN [23]:** This is a neighborhood based collaborative filtering algorithm based on items. The idea is to compute an item-item similarity matrix based on Pearson correlation coefficient.
- **ImplicitMF [9]:** This method was proposed to deal with implicit feedback data when using Matrix Factorization algorithm. They added a weight term to consider the confidence of an item and proposed an alternating least square algorithm to learn user's and item's latent vectors.
- **BPRMF [21]:** This method tailored for implicit feedback is a Matrix Factorization based method, but rather than minimizing the predicted "rating" as done in classical MF, it minimizes the pairwise ranking loss.
- **FM [20]:** This method was proposed by Rendle in order to incorporate contextual data in the recommendation. Instead of computing only users' and items' latent vectors, it computes also features latent vectors.
- **WDL [2]:** Wide & Deep Learning model combined a deep component (similar to the deep component used in our model) plus a wide component which can be seen as a linear model that computes cross products between input features. For this baseline, we used only the deep component. Indeed, The wide component is not adapted for our case as it requires the impression items.
- **NCF [8]:** Neural Collaborative Filtering is a state-of-the-art collaborative filtering approach. It combines the (user, item) interaction as input of a multilayer perceptron and a single layer perceptron that models the matrix factorization method.
- **NFM [7]:** Neural Factorization machines is a state-of-the-art model for context-aware recommendation. The factorization component used in our model represents a part of the neural factorization machines, the other part is a multilayer perceptron to which we add the linear term of factorization machines formula.

Implementation Framework & Parameter Settings: Our model plus all the baselines were implemented using Python and Tensorflow library[9]. The hyper-parameters of all the models were tuned using grid-search algorithm. First, we initialized all the weights randomly with a Gaussian Distribution ($\mu = 0$, $\sigma = 0.01$), and we used mini-batch Adam optimizer [11]. It is worth mentioning that other optimizers could be used in order to minimize the loss function defined in (6), however, Adam Optimizer has shown to be the most efficient in time and also accuracy. We evaluated our model using different values for hyper-parameters: the size of traveler and city embedding layers: $E\_size \in \{32, 64, 128\}$, the features vector size of the factorization machines component: $k \in \{8, 16, 32, 64\}$, the batch size: $B\_size \in \{64, 128, 256, 512, 1024\}$, the number of epochs: $epochs \in \{5, 10, 15, 20\}$ and the learning rate: $l_r \in \{0.001, 0.005, 0.006, 0.008, 0.1\}$.

5.2 Results and Discussion

Deep component performance: In the table 6, we present the results of the deep component using the metrics we have defined in the previous subsection. We can notice that the traveler demographics information (DCU) remarkably improved the performance of deep component that has only the traveler and city embedding as input (DC). The scores of HR@10 and MRR@10 increased by 15%. As for the city embeddings, we can notice that using the textual embeddings (DCTEI) improved the results by 9% and 6%. When considering the traveler demographics in addition to the textual city embedding (DCUTEI) the results improved by 27% for HR@10 and 28% for MRR@10. Finally, when concatenating all the deep component input, we improved our scores by 30% and 28%. The batch size used is 256, the number of epochs used is 8. Even if the loss defined in equation 6 decreased after 5 epochs for both the training and validation set, the neural network is over-fitting and the metrics HR@10 and MRR@10 decreased. Finally we used 0.006 as learning rate and 128 as the size of the traveler and city embedding layers. It is worth to notice that the hidden layer size

[9]Python Tensorflow API: https://www.tensorflow.org

Table 6: Contribution of each input in the deep component

Model	HR@10	MRR@10	#Layers	1st Layer size
DC	0.66	0.32	2	256
DCTEI	0.72	0.34	2	512
DCKGEI	0.79	0.36	2	512
DCU	0.76	0.37	2	512
DCUTEI	0.84	0.41	2	1024
DCUKGEI	0.84	0.40	2	1024
DCUI	**0.86**	**0.41**	3	1024

Table 7: Contribution of each input in the deep knowledge factorization machines

Model	HR@10	MRR@10	#Layers	1st Layer size
DKFM_CTXT	0.72	0.34	2	256
DKFMTEI	0.79	0.37	2	512
DKFMKGEI	0.80	0.38	2	512
DKFMU	0.82	0.38	2	512
DKFMUTEI	0.84	0.41	2	1024
DKFMUKGEI	0.85	0.42	2	1024
DKFM	**0.88**	**0.44**	3	1024

and the number of layers used vary over the input size as shown in the table 6.

Factorization machines contribution: The deep knowledge factorization machines was implemented by combining the feature vectors obtained from the factorization component by using the number of passengers in the booking in addition to the departure day of week as contextual data, and the deep component. Similarly to the table 6, we report the contribution of each input in the table 7. It is worth to notice that adding the contextual data increase the score of HR@10 and MRR@10 (DKFM_CTXT).

HR@10 and MRR@10 increased by 10% and 9% respectively when adding the textual embeddings (DKFMTEI). As for the knowledge graph embeddings, both scores increased by 11% (DKFMGEI). When considering the user demographics data (DKFMU), one can notice that the results improved by 13% and 11%. Finally, when considering all the input of our model DKFM, the results were 22% for HR@10 and 30% for MRR@10 better than the DKFM with only contextual data. We tested different values of k from 8 to 128, and we did not notice any change neither on the test loss, nor on the metrics HR@10 and MRR@10.

DKFM against baseline Models: We have measured HR@10 and MRR@10, for the different baseline models implemented and for our model DKFM. We present the results in Figure 4. As shown in Figure 4, our model outperforms the collaborative filtering methods demonstrating the importance of adding the city embeddings, traveler demographics data and the contextual data. It also shows a slight improvement over wide and deep learning and factorization machines where one is using city embeddings and traveler demographics data and the other is using contextual data. Considering that training time is also an important aspect to consider when doing recommendation, we measured training times for both DKFM and WDL models. For each epoch, the training time is equal to **24** seconds for DKFM and **15** seconds for WDL. For our experiments, we used an NVIDIA Tesla K40C GPU with 12 GB of memory. Finally we compute the metrics HR@K and MRR@K for different values of K and we report the results in figure 5.

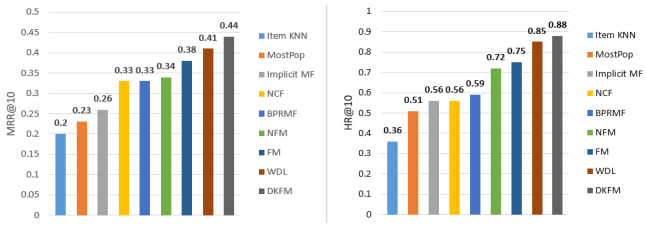

Figure 4: HR@10 & MRR@10

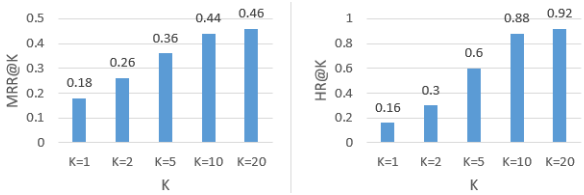

Figure 5: HR@K & MRR@K for DKFM

6 CONCLUSIONS AND FUTURE WORK

In this work, we have presented a neural network based approach to predict next trip destination for travelers. We have leveraged two external resources in order to enrich the characteristics of our destinations represented by a city. Our recommender system could be used for offline recommendations by sending emails to travellers, as well as for online recommendations on the airline's website to inspire them. We publish our code as open source in order to ease reproducibility: https://gitlab.eurecom.fr/amadeus/DKFM-recommendation. We conducted several experiments to address our research questions:

RQ1: What is the contribution of the deep component? The results of the experiments presented in table 6 show that when using Deep component with traveler demographics and city embeddings enhance significantly the next trip recommendation. Indeed, as shown in Figure 4, WDL outperforms all the collaborative filtering methods and also factorization machines model.

RQ2: What is the contribution for each input used in the deep component: traveler demographics data, city embeddings? The experiments demonstrated that the traveler demographics data improves remarkably the performance of the deep component. The textual embedding also improves the performanc of the deep component, but less than the traveler demographics. Finally, it showed that the knowledge graph embeddings improve more the score of the metrics in comparison with the two other inputs. Feeding DKFM with more relevant features improve the results by allowing the Deep Neural Network to learn more associations by combining the input features. However our model is considered as a black box

Table 8: Tuned Hyper-parameter for DKFM

Hyper-parameter	E_size	l_r	k	B_size	$epochs$
Value	128	0.06	8	256	8

in the sense that we are not able to say which input features were most important. This is in contrast with a regression or decision tree model where the information gain gives the information about the important features.

RQ3: What is the contribution of factorization machines component? Factorization machines component showed better results than collaborative filtering methods. However, the classical factorization machines got better results. This can be explained by the linear term that we do not use in our model. Indeed this term models the strength of each features.

RQ4: How our model perform in comparison with the baseline models defined later in this section? Our model outperforms all the collaborative filtering methods plus the deep learning methods [2, 7]. One can notice that the baseline MostPop has a good score for HR@10. Indeed, as there are only 119 cities to recommend (a few number of items), recommending the 10 most popular cities is at least one time out of two relevant.

RQ5: How the performance of our model is affected by the hyper-parameters? We ran grid-search on all the DKFM's hyper-parameters. The tuned hyper-parameter for our dataset are presented in table 8. We compared range of values for the size of the embedding layers, where 128 showed to be the value that has the highest HR@10, and we also compared different values for k: the size of feature vector from factorization machines component. For all the values we compared (see Section 5.2), HR@10 and MRR@10 did not vary.

In the future, we will first explore new data sources such as images that would help to enrich cities' characteristics. Next, we will investigate how we could improve the performance of our DKFM model by exploring new loss functions such as pairwise loss used in BPRMF [21]. We will also experiment the use of similarity measures inside the neural network such as cosine similarity. Finally, we plan to evaluate the performance of our recommender system for business travels.

REFERENCES

[1] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *26th International Conference on Neural Information Processing Systems - Volume 2*. Curran Associates Inc., USA, 2787–2795.

[2] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. In *1st Workshop on Deep Learning for Recommender Systems*. ACM, New York, NY, USA, 7–10.

[3] Thierry Delahaye, Rodrigo Acuna-Agost, Nicolas Bondoux, Anh-Quan Nguyen, and Mourad Boudia. 2017. Data-driven models for itinerary preferences of air travelers and application for dynamic pricing optimization. *Journal of Revenue and Pricing Management* 16, 6 (01 Dec 2017), 621–639. https://doi.org/10.1057/s41272-017-0095-z

[4] Gintare Karolina Dziugaite and Daniel M. Roy. 2015. Neural Network Matrix Factorization. *CoRR* abs/1511.06443 (2015).

[5] Aditya Grover and Jure Leskovec. 2016. Node2Vec: Scalable Feature Learning for Networks. In *22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, USA, 855–864.

[6] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. 1725–1731.

[7] Xiangnan He and Tat-Seng Chua. 2017. Neural Factorization Machines for Sparse Predictive Analytics. In *40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, USA, 355–364.

[8] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 173–182.

[9] Y. Hu, Y. Koren, and C. Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE Computer Society, Washington, DC, USA, 263–272.

[10] Houda Khrouf and Raphaël Troncy. 2013. Hybrid Event Recommendation Using Linked Data and User Diversity. In *7th ACM Conference on Recommender Systems*. ACM, Hong Kong, China, 185–192.

[11] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014).

[12] Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. 2015. Skip-Thought Vectors. In *NIPS*.

[13] Julia Kiseleva, Melanie J.I. Mueller, Lucas Bernardi, Chad Davis, Ivan Kovacek, Mats Stafseng Einarsen, Jaap Kamps, Alexander Tuzhilin, and Djoerd Hiemstra. 2015. Where to Go on Your Next Trip?: Optimizing Travel Destinations Based on User Preferences. In *38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, USA, 1097–1100.

[14] Lajanugen Logeswaran and Honglak Lee. 2018. An efficient framework for learning sentence representations. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJvJXZb0W

[15] Augusto Q. Macedo, Leandro B. Marinho, and Rodrygo L.T. Santos. 2015. Context-Aware Event Recommendation in Event-based Social Networks. In *9th ACM Conference on Recommender Systems*. ACM, New York, NY, USA, 123–130.

[16] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in Pre-Training Distributed Word Representations. In *International Conference on Language Resources and Evaluation (LREC 2018)*.

[17] Diego Monti, Enrico Palumbo, Giuseppe Rizzo, Raphaël Troncy, and Maurizio Morisio. 2018. Semantic Trails of City Explorations: How Do We Live a City. *CoRR* abs/1812.04367 (2018).

[18] Enrico Palumbo, Giuseppe Rizzo, and Raphaël Troncy. 2017. Entity2Rec: Learning User-Item Relatedness from Knowledge Graphs for Top-N Item Recommendation. In *Eleventh ACM Conference on Recommender Systems*. ACM, New York, NY, USA, 32–36.

[19] Enrico Palumbo, Giuseppe Rizzo, Raphaël Troncy, Elena Baralis, Michele Osella, and Enrico Ferro. 2018. Translational models for item recommendation. In *ESWC 2018, 15th European Semantic Web Conference*. Heraklion, GREECE.

[20] Steffen Rendle. 2010. Factorization Machines. In *2010 IEEE International Conference on Data Mining*. IEEE Computer Society, Washington, DC, USA, 995–1000.

[21] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, Arlington, Virginia, United States, 452–461.

[22] Francesco Ricci. 2002. Travel recommender systems. *IEEE Intelligent Systems* 17, 6 (2002), 55–57.

[23] Badrul Sarwar, George Karypis, Joseph A Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *10th International Conference on World Wide Web, WWW 2001*. ACM, Inc, New York, NY, USA, 285–295.

[24] Zhu Sun, Jie Yang, Jie Zhang, Alessandro Bozzon, Long-Kai Huang, and Chi Xu. 2018. Recurrent Knowledge Graph Embedding for Effective Recommendation. In *12th ACM Conference on Recommender Systems*. ACM, New York, NY, USA, 297–305.

[25] Q. Wang, Z. Mao, B. Wang, and L. Guo. 2017. Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE Transactions on Knowledge and Data Engineering* 29, 12 (2017), 2724–2743.

[26] Wolfgang Wörndl, Alexander Hefele, and Daniel Herzog. 2017. Recommending a sequence of interesting places for tourist trips. *Information Technology & Tourism* 17, 1 (2017), 31–54.

[27] Carl Yang, Lanxiao Bai, Chao Zhang, Quan Yuan, and Jiawei Han. 2017. Bridging Collaborative Filtering and Semi-Supervised Learning: A Neural Approach for POI Recommendation. In *23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, USA, 1245–1254.

[28] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative Knowledge Base Embedding for Recommender Systems. In *22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, New York, NY, USA, 353–362.