Predicting Your Next Stop-over from Location-based Social Network Data with Recurrent Neural Networks

Enrico Palumbo
ISMB
Turin, Italy
EURECOM
Sophia Antipolis, France
palumbo@ismb.it

Giuseppe Rizzo
ISMB
Turin, Italy
giuseppe.rizzo@ismb.it

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

Elena Baralis
Politecnico di Torino
Torino, Italy
elena.baralis@polito.it

ABSTRACT

In the past years, Location-based Social Network (LBSN) data have strongly fostered a data-driven approach to the recommendation of Points of Interest (POIs) in the tourism domain. However, an important aspect that is often not taken into account by current approaches is the temporal correlations among POI categories in tourist paths. In this work, we collect data from Foursquare, we extract timed paths of POI categories from sequences of temporally neighboring check-ins and we use a Recurrent Neural Network (RNN) to learn to generate new paths by training it to predict observed paths. As a further step, we cluster the data considering users' demographics and learn separate models for each category of users. The evaluation shows the effectiveness of the proposed approach in predicting paths in terms of model perplexity on the test set.

KEYWORDS

Sequence learning, path recommendation, tourism, POI recommendation

1 INTRODUCTION

Location-based Social Networks (LBSN) allow users to *check-in* in a Point-of-Interest (POI)[1] and share their activities with friends, providing publicly available data about their behavior. One of the distinctive features of LBSN data with respect to traditional location prediction systems, which are mainly based on GPS data and focus on physical mobility [32], is the rich categorization of POIs in consistent taxonomies, which attribute an explicit semantic meaning to users' activities. The availability of venue categories has opened new research lines, such as statistical studies of venues peculiarities [17], automatic creation of representations of city neighborhoods and users [22, 24], definition of semantic similarities between cities [23]. Most importantly, venue categories play an important role in POI recommender systems, as they enable to model user interests and personalize the recommendations [18]. In the past years, little attention has been dedicated to the temporal correlations among venue categories in the exploration of a city, which is nonetheless a crucial factor in recommending POIs. Consider the example of a check-in in an *Irish Pub* at 8 PM: is the user more likely to continue her evening in a *Karaoke Bar* or in an *Opera House*? Better a *Chinese Restaurant* or an *Italian Restaurant* for dinner after a *City Park* in the morning and a *History Museum* in the afternoon? Note that predicting these sequences require an implicit modeling of at least two dimensions: 1) temporal, as certain types of venues are more temporally related than others (e.g. after an *Irish Pub*, people are more likely to go to *Karaoke* than to a *History Museum* 2) personal, as venue categories implicitly define a user profile, independently from their order (e.g. *Steakhouse* and *Vegetarian Restaurant* do not go frequently together). Most of existing studies attempt to model directly sequences of POIs rather than their categories to recommend the next POI to a user (see 'next POI prediction' in Sec. 2). In this work, we focus on modeling sequences of POI categories to enhance the generality and the portability of the obtained results. This can be considered as a first step in the next POI prediction problem, as the POI category can then be turned into a specific POI by querying a database of POIs according to a variety of parameters, such as the user context (e.g. position, weather) and/or specific POI features such as popularity, average prices and the like. In order to address this problem, we first collect users' check-ins from Foursquare and extract their corresponding venue

[1]The term venue is used interchangeably with POI in this work to describe an entity that has a somewhat fixed and physical extension as defined by http://schema.org/Place

categories, segmenting them into a set of temporally neighboring activities, which we call *paths*. Then, we train a Recurrent Neural Network to learn to predict these paths in order to generate new ones, thanks to its architecture that is specifically meant to model temporal sequences without specifying a specific memory length. In the attempt to take into consideration the fact that the nature of the generated sequences is not universal, but it critically depends on the typology of user, we cluster users in groups and learn separate models for each of them. Differently from previous work [31], we cluster users based on their demographics rather than on their past activities, consistently with the intent of obtaining results that are portable to new data without a cold start problem.

The main scientific contributions of this paper are: (1) addressing the problem of next POI category using a machine learning approach on sequences of temporally consecutive check-ins; (2) use of a Recurrent Neural Network (RNN) with Gated Recurrent Units (GRU) with multiple layers as a model; (3) an initialization of the vectors fed to the neural networks using an unsupervised feature learning algorithm (node2vec) on the hierarchical graph modelling the Foursquare taxonomy; (4) a user clustering based on demographics that is not affected by the cold start problem; (5) an evaluation protocol based on perplexity, which is new in this domain and is able to address the limitations of using accuracy on a set of interdependent target categories.

2 RELATED WORK

Venue categories

The availability of venue categories from LBSN data has inspired a number of studies in the past years. In [17], the authors assess the correlations among venue categories and popularity with a statistical study on a large sample of check-ins collected from different geographical regions. In [25], the authors leverage venue categories to automatically create a high level map of the neighborhoods of a city using density-based clustering techniques. In [22], the authors use venue categories to create semantic representation of city neighborhoods and users. In [23], the authors create a semantic representation of a city as a bag of venue categories and use it to define a similarity measure between cities.

Next POI recommendation

All of these studies, however, do not take into account the temporal dependence among venue categories, i.e. they do not attempt to predict where a user will move next considering the history of her movements in terms of venue categories. This task is similar to the next POI prediction, which has received some attention in the past years. For instance, in [4] the authors propose a matrix factorization method including personalization and geographic constraints that attempts to predict the next check-in of the user based on her past activities and geographical factors. In [8], the authors use a metric embedding approach to develop a personalized model of the sequential transition of POIs. These two studies directly develop a model to recommend the next POI, while, similarly to our approach, in [31] the authors focus on modeling sequences of venue categories. They propose a framework that uses a mixed hidden Markov model to predict the most likely next venue category and recommend POIs belonging to the most likely next category in the neighborhood of the user. Although this work has some common features with the one proposed in this paper, such as the modeling of venue categories transitions rather than directly the POIs transitions, there are important differences. First, they utilize Gowalla's[2] data and venue categorization, which included only nine broad categories, such as *Food* or *Shopping*, which can reasonably considered independent among each other. Our work, on the other hand, is based on Foursquare taxonomy, which includes 920 categories organized in a hierarchical fashion, and thus requiring a more complex modeling effort. Secondly, while they cluster users based on their past activities, we follow a different approach, considering user demographics, effectively tackling the new user problem. Third, they use a Hidden Markov Model while we use an approach based on Recurrent Neural Networks. Finally, they evaluate the proposed approach using accuracy, while we use perplexity.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) have received a great deal of attention in machine learning research lately [16], as, thanks to their improved architectures [5, 12] and the advancements in computational power, they are able to effectively model sequences. For this reason, they have been used successfully for tasks such as speech recognition [10], sentiment analysis [29], image captioning [13] and neural language models [20]. One of the typical applications of RNN in the field of language modeling is that of generating text by recursively predicting the next word in a sentence [28]. This task is very similar to the use of RNNs in this work, in which the analogy is that of interpreting a sequence of venue categories coming from users' check-ins as a sentence in a text.

[2] https://en.wikipedia.org/wiki/Gowalla

Itinerary recommendation

The problem of modeling and recommending paths to users share important features with that of itinerary recommendation, which aims at recommending sequences of POIs, considering constraints such as time, budget and personal preferences. Typically, to each POI a score is assigned based on popularity and/or personal preferences, travel times between POIs are inferred from data, and the problem of itinerary recommendation is tackled as an optimization problem where the objective is to maximize the total possible score of the itinerary while complying with the constraints [7, 15, 30]. Note that the greatest difference with our approach is that we do not explicitly formulate the optimization problem with constraints, but rather assume that good paths will be learned from LBSN data.

3 APPROACH

Problem statement

In this work, we address the problem of next POI category prediction, i.e. we aim to learn to predict the category of the next POI that a user will visit, in order to be able to generate and recommend new paths.

Definition 1. *Given the space of POI categories C, the space of check-in ids I, the space of timestamps T, the space of users U, a check-in is a set $v = \{i, c, \tau, u\}$ where $i \in I$ is the check-in id, $c \in C$ is the category of the POI, $\tau \in T$ is the timestamp at which the check-in has been performed and $u \in U$ is the user who has performed the check-in.*

Definition 2. *A path is an ordered sequence of POI categories $(c_1, c_2, ..., c_t)$, extracted from a sequence of temporally ordered check-ins performed by a particular user $u \in U$, i.e. $(\{i_i, c_i, \tau_i, u\})$ for $i = 1..N$ and where $\tau(i+1) > \tau(i)\ \forall i$.*

Definition 3. *We define a category index $\alpha \in \mathbb{N}$ with $\alpha = 1..|C|$ and that uniquely identifies a category $c^\alpha \in C$.*

In order to learn to generate the next category c_{t+1} of a path, we collect M paths from LBSN and learn a model of the conditional probability $P(c_{t+1}|c_t, c_{t-1}, c_{t-2}, ..., c_1)$ from these sequences of POI categories. Then, from this model, the next category c_{t+1} can be determined as:

$$c_{t+1} = \arg\max_{c \in C} P(c|c_t, c_{t-1}, c_{t-2}, ..., c_1) \tag{1}$$

Model

We propose an approach based on Recurrent Neural Networks, which are specifically meant to deal with sequential data. The main difference of RNNs with respect to standard feed-forward neural networks is the presence of a hidden state variable h_t, whose value depends both on the input data presented at time x_t and, by means of loop connections, on the previous hidden state h_{t-1}[9]. A typical application of RNNs in neural language modeling is that of generating text recursively applying a "next word prediction" [28], and in the same spirit we address the problem of next POI category prediction. The main idea is that of using a supervised learning approach where the targets correspond to the inputs shifted in time, i.e. $X = \{({c^j}_0, {c^j}_1, ..., c^j_{N_j-1})\}$ and $Y = \{({c^j}_1, {c^j}_2, ..., c^j_{N_j})\}$ where $j = 1...M$ is the path index and N_j is the length of the $j-th$ path. The architecture of the neural network is illustrated in Fig. 1. To simplify the notation, we now drop the path index j and consider one path to illustrate the functioning of the network. A venue category c_t is fed into the network via an encoding into an input vector x_t, which is then passed to a Gated Recurrent Unit. Gated Recurrent Units (GRU) are gating mechanisms that improve the ability of the RNNs to store long sequences and that recently have been proven to be as effective as more complicated architectures such as Long Short-Term Memory (LSTM) units [5]. The update of the GRU unit hidden state, i.e. the computation of the new state h_t given the previous state h_{t-1} and the current input x_t, is described by the following equations:

$$r_t = sigmoid(W_r h_{t-1} + W_r x_t + b_r) \tag{2}$$

$$h'_t = tanh(W_i(r_t \otimes h_{t-1}) + W_i x_t + b_i) \tag{3}$$

$$z_t = sigmoid(W_z h_{t-1} + W_z x_t + b_z) \tag{4}$$

$$h_t = z_t \otimes h' + (1 - z_t) \otimes h_{t-1} \tag{5}$$

where *sigmoid* and *tanh* indicate respectively the sigmoid and hyperbolic tangent activation functions and $\otimes$ represent the element-wise product of the matrices. r is called the 'reset gate' and it allows to forget or remember the previous state h_{t-1} when generating the candidate state h'_t. z is called the 'update gate' and intuitively it controls how much the unit needs to update its state. W_i, W_r, W_z are weight matrices that are learned during the training.

The GRU computes the hidden state h_t which is stored for the next iteration and used to compute the output of the current iteration o_t. Before computing the output o_t, during training time, a Dropout layer is applied. The Dropout layer is a regularization mechanism which, at training time, randomly switches off a fraction p of neurons, called the *dropout rate*, preventing them from co-adapting and overfitting the sampled data [27]. Dropout can be modelled with a mask vector m_t, whose values can be either 1 or 0 with probability p. After the dropout layer, the output state $o_t = tanh(W_o h_t m_t)$ is computed using a fully connected layer whose weights are defined by the matrix W_o, which is learned at training time. W_o is shaped so that the dimension of the output vector is equal to the number of possible categories, i.e. $|o_t| = |C|$. Thus, we can index the components of the output vector

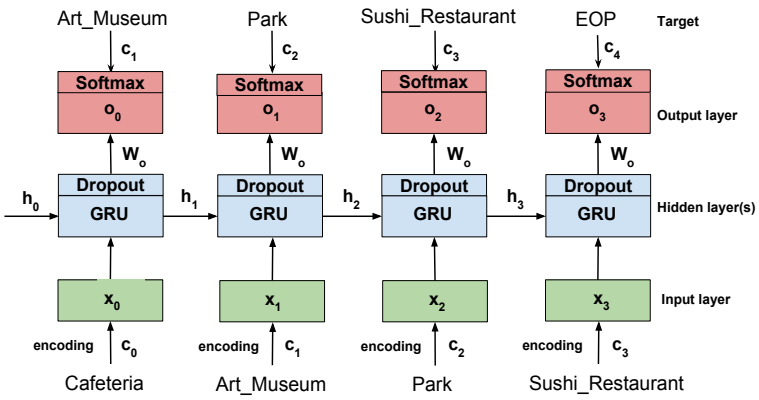

Figure 1: Architecture of the RNN. 'EOP' is a special symbol describing the end of a path.

using the category index o_t^α. Then, the Softmax layer normalizes the outputs, turning them into a probability distribution over a set of possible outcomes [2]:

$$softmax(o_t^\alpha) = \frac{e^{o_t^\alpha}}{\sum_{k=1}^{|C|} e^{o_t^k}} \tag{6}$$

In this way, the Softmax layer models the probability distribution of the next category:

$$softmax(o_t^\alpha) = P(c_{t+1} = c^\alpha | c_t, c_{t-1}, c_{t-2}, ..., c_1) \tag{7}$$

as o_t^i depends on the current category encoding x_t of c_t, but also on all the previous encodings of the sequence by means of the hidden state h_t. During the training process, we train the network to produce a probability distribution of categories that is as close as possible to that observed in the data, i.e. maximizing the probability of the observed data. Therefore, we define the loss L_t as the cross entropy:

$$L_t = -logP(c_{t+1} = c_{t+1}^\alpha | c_t, c_{t-1}, c_{t-2}, ..., c_1) = -log(softmax(o_t^\alpha)) \tag{8}$$

where c_{t+1}^α is the category observed in the data as $t+1$ element of the path. The loss is optimized using Adam [14], an enhanced version of the stochastic gradient descent that introduces momentum and adaptive learning rates. The gradients of the loss function are computed using back propagation on the unrolled neural network [26]. The model has a number of hyper-parameters, such as the number of neurons in the hidden state n_{hidden}, the number of hidden layers n_{layers}, the learning rate l_r and the number of epochs η. We optimize these hyper parameters using a grid search on a validation set (see Sec. 5).

Feature learning from category hierarchy

In this work, the space of possible categories C is defined by the Foursquare Taxonomy[3], which defines and classifies categories in a hierarchical ontology. As can be seen in Fig. 1, it is necessary to specify an encoding to turn the categories into input vectors to be fed into the neural network. A simple and widespread approach to encode categorical variables is that of using the so called *one-hot* encoding, i.e. to use a binary vector whose dimension is equal to the size of the vocabulary $d = |C|$ where only one component is different from 0 using the category index α:

$$x_k^{one-hot}(c^\alpha) = \begin{cases} 1 \iff k = \alpha \\ 0 \iff k \neq \alpha \end{cases}$$

The one-hot encoding is a sparse representation and, although straight forward and intuitive, has a number of shortcoming. First, the size of the input vector depends on the size of the category vocabulary C. This can be defined as the total number of distinct categories that appear in the data, hindering the applicability of the model to unobserved categories, or as the total number of possible categories, which can result in a waste of computational resources when certain categories are not observed in the data. Secondly, the one-hot encoding considers each category as independent from each other and equidistant from the others. Consider as an example the case of three categories: *Restaurant* = (1,0,0), *Italian Restaurant* = (0,1,0), *Movie Theater* = (0,0,1). This representation does not allow to determine whether *Restaurant* is more similar to *Italian Restaurant* or *Movie Theater* and thus fails to effectively represent the hierarchical relations among categories. For this reason, we use node2vec [11], an unsupervised feature learning approach that maps nodes in a graph to a dense vector representation in a Euclidean space of fixed dimension preserving the structure of the graph. Node2vec can be seen as an adaptation of the word2vec model [21] to graphs, as it simulates a random walk on the graph, turning it into an order sequence of nodes, which constitutes the "words" of a document that is then processed using word2vec. In node2vec, we use a uniform exploration,

[3] https://developer.foursquare.com/categorytree

i.e. $p = q = 1$, a dimension of the obtained vector $d = 100$ and 100 walks per category node. In Sec. 5, we compare the results obtained with $x^{one-hot}$ and $x^{node2vec}$, showing that the latter leads to better result and faster computation. A dynamic visualization of the node2vec category embedding can be found online[4]. To obtain a good visualization, we suggest to use TSNE [19] with at least perplexity 20 and 1000 iterations.

Personalized model

In order to take into account the fact that the next POI category prediction problem can strongly be influenced by personal attributes of a user, we segment the set of users U in a collection of clusters according to the user demographics. Considering the set of languages $l \in L$ and of genders $g \in G$, we generate all possible clusters U_{lg}, U_l, U_g and we segment the whole set of paths P accordingly generating P_{lg}, P_l, P_g. We split each of them into training set and test set containing respectively 80-20% of the data, we train the model on the training sets and assess performance on the test sets.

4 EXPERIMENTAL SETUP

Data Collection

In order to collect check-ins and sufficient user information to perform a demographic clustering, we use as data sources both the Twitter and the Foursquare API. We collect via the Twitter Search API check-ins done through the Swarmapp[5] application and publicly posted on Twitter, obtaining 1.5M check-ins from 235.6K users in the temporal interval going from 05-04-2017 to 11-04-2017. From this data, we are able to extract for each check-in the language spoken by the user and the id of the check-in. With the check-in id, to gather additional information about the venue and the user, we query the Foursquare API[6], obtaining the venue category c and the user gender. Thus, for each user, we have: $(user\_id, language, gender, (c_1, \tau_1), ..., (c_N, \tau_N))$, where τ is the timestamp of the check-in.

Preprocessing and Path Extraction

Among all users, only a small part of them uses the application frequently enough to be likely to generate a path. Thus, as a pre-filter to speed up the next processing steps, we filter out users with less than 10 check-ins. We also observe that there are users with a very large number of check-ins, who are likely to be bots. In order to remove them, we develop a heuristic according to which if a user has done more than twice two check-ins in one minute is a bot. By manually checking the results on a sample of 50 users, we observe no false positives. Then, in order to extract the paths, i.e. temporal sequences of correlated venue categories, we apply the principle according to which two check-ins are part of the same path if and only if they both occur within a time window, similarly to what has been done in [7]. Thus, given a set of timestamped check-ins performed by a given user $(c_1, \tau_1), ..., (c_N, \tau_N)$, we split this sequence in multiple paths whenever $\tau_{i+1} - \tau_i > 8h$, i.e. the time difference between two consecutive check-ins is higher than 8 hours. Isolated check-ins, i.e. with $\tau_{i+1} - \tau_i > 8h$ and $\tau_{i-1} - \tau_i > 8h$, are removed from the data. We obtain 29.5K paths, with an average length of 4.2 and a maximum length of 50. In Tab. 1, we report the number of users and check-ins after each preprocessing step.

	Users	Check-ins
Collection	235.6K	1.5M
More than 10 check-ins	19.5K	400K
Bot removal	12.4K	184K
Path extraction	12K	123K

Table 1: Number of distinct users and check-ins originally and after each preprocessing step.

User clustering

As we have mentioned in Sec. 1, the way in which tourists explore a city is different and personalized. Although a personalized path recommendation would be desirable, the amount of training data is not sufficient to achieve such a goal. Therefore, we cluster users in groups and tailor the path recommendation to a given user group. In order to obtain results that are general and do not depend on the specific dataset that we have collected, we propose a clustering approach based on the user demographics information that we have collected, i.e. the language and the gender of the user, segmenting the collected paths according to these clusters. We observe 30 distinct languages in the data and count the number of paths per each language-gender pair. We also consider higher level clusters such as: (all, gender) and (language, all), which can be used when only one of the two features about the user is available. We require to have at least 100 users to create a cluster, obtaining 22 distinct clusters.

Evaluation

In the experimental part of this work, we try to answer to the following research questions:
1) What is the most effective architecture of the RNN model,

[4]http://projector.tensorflow.org/?config=https://gist.githubusercontent.com/enricopal/9a2683de61f5fb16c4f59ae295e3fef7/raw/159df72f47e881d0f314096fcc8ea561fb7132b9/projector_config.json
[5]https://www.swarmapp.com
[6]https://developer.foursquare.com/docs/checkins/resolve

i.e. what are the best hyper-parameters of the model?
2) Is the dense encoding provided by node2vec more effective than the sparse one-hot encoding?
3) Are Recurrent Neural Networks better at generating paths with respect to a model with a fixed memory window, such as a bigram model?
4) Is the clustering of users favoring or hindering the effectiveness of the model?
In order to answer to these questions, we need to define an appropriate metric to measure the performance of the model. Although accuracy has a straight forward interpretation as it is simply measured as the fraction of correctly predicted venue categories, it would consider all categories independently and weight all errors in the same way. For example, predicting *Sardinian Restaurant* or *Thai Restaurant* when the true category is *Roman Restaurant* would count as an error in the same way. Thus, we opt for a different metric, commonly used in neural language modeling evaluation, that is perplexity [1, 20]. Perplexity is defined as the exponential of the average negative log-likelihood of the model, which in our case becomes:

$$ppl = 2^{-\frac{1}{(\sum_{k=1}^{M} N_k)} \sum_{j=1}^{M} \sum_{t=1}^{N_j} log P(c_{t+1}^j = c_{t+1}^\alpha | c_t^j .. c_1^j)} \quad (9)$$

where M is the total number of paths, N_k is the length of the k-th path and c_{t+1}^α is the category observed in the data as $t+1$ element of the path. Intuitively, perplexity measures the "surprise" of the model in observing the test data. Note that if we roll an ideal die with a number of faces equal to the number of categories C, i.e. $p = \frac{1}{|C|}$, the perplexity is then exactly $ppl = |C|$. Thus, the perplexity can be interpreted as the number of possible outcomes among which a random system should guess. The lower the perplexity, the better is the model. Also note that ppl is equal to the exponential in base 2 of the cross entropy between the model and the data distribution, i.e. the average of the loss L_t over all timesteps of all paths and is thus naturally optimized by the model training. Having defined a metric to evaluate the model, we create a training set and a test set, containing respectively 80-20% of the paths. The validation set used for the optimization of the hyper-parameters is in turn extracted as a 20% of the training set.

5 RESULTS

Hyper parameters optimization

In order to optimize the hyper parameters (experiment 1), we perform a grid search, i.e. we explore all the possible combinations of the following values:
number of neurons in the hidden layer: $n_{hidden} = [64, 128]$
learning rate: $l_r = [10^{-4}, 5 * 10^{-4}, 10^{-3}]$
number of epochs: $epochs = [1, 2, 5, 10]$
number of hidden layers: $n_{layers} = [2, 3]$

rank	n_hidden	l_r	epochs	n_layers	ppl
1	64	10^{-4}	5	3	71.333
2	64	10^{-4}	5	2	71.609
3	64	10^{-4}	2	3	71.630
4	128	10^{-4}	2	2	71.645
5	128	10^{-4}	5	2	72.048

Table 2: Perplexity on validation set for the top 5 configuration of hyper parameters.

For each configuration $(n_{hidden}, l_r, epochs, n_{layers})$, we train the model and measure its perplexity on validation data, exploring a total of 48 possible configurations. In Tab. 2, we report the best 5 configurations. We can observe that a small learning rate is helping the model learn and that depth, i.e. number of hidden layers, is more effective than width, i.e. number of neurons in the hidden layers. We also observe that training the model for more epochs increases the performance.
In the rest of the section, unless otherwise specified, we use the best configuration of the model.

Test set scores

We now show the results corresponding to the experiments 2 and 3, i.e. we compare the model initialized with node2vec vectors, with one-hot vectors and the baseline bigram model on the test set. The bigram model is built by estimating the 1-st order transition probabilities $P(c_{t+1}|c_t)$ by counting their normalized co-occurrence frequencies on training data and using add-one smoothing to account for bigrams that do not appear in the training data [3]:

$$P(c_{t+1}|c_t) = \frac{max(1, v_{c_{t+1}, c_t})}{|C| + v_{c_t}} \quad (10)$$

where v_{c_{t+1}, c_t} denotes the frequency of the bigram (c_{t+1}, c_t), i.e. of co-occurrence of the categories c_{t+1} and c_t whereas v_{c_t} denotes the frequency of the category c_t.
The results are reported in Tab. 3. We can observe that the RNN with node2vec initialization performs better with respect to the other systems and that RNN with one-hot encoding is still far better than the bigram model. This shows, on the one hand, the effectiveness of node2vec as an initialization strategy and that of RNNs in predicting paths. We also observe that the difference between the node2vec and one-hot initialization is small, highlighting the ability of RNNs of learning well also starting from a sparse representation. However, we observe a ratio in computing time of 1.35, as the model runs in 39 minutes with node2vec embeddings and in 53 minutes with one-hot encoding on a server with 48 CPU cores and 256GB of RAM, thanks to the 'compressed' representation of the inputs. In general, we can say that the

System	ppl
RNN-node2vec	75.271
RNN-onehot	76.472
bigram+smoothing	125.361
random	741

Table 3: Perplexity on test set for the proposed approach and baselines.

proposed approach achieves a perplexity of 75.271 on the test set, shrinking of about 10 times the space of possible categories among which a random system would have to guess.

Personalized model

In this section, we compare the performance of the global model to that of the model trained on the paths belonging to a specific user cluster (experiment 4). The perplexity score, the number of paths, the average path length and the max length are reported for each user cluster in Tab. 4. Note that not all users choose to show their gender or language, as can be verified for example by noting that $Users(M) + Users(F) < Users(All)$. From the results, we can observe that the model perplexity is increasing for certain user clusters and decreasing for others, hinting to the fact that some user clusters might be less predictable than others. For instance, we observe that Turkish speaking users have a much lower perplexity than Dutch speaking users and we might be tempted to conclude that the behavior of the former category is much easier to predict than that of the latter. However, an important role is played by the dimension of the training set, which varies significantly across the clusters and that is a key ingredient in the learning process. In order to look into the correlation between the model perplexity and the number of paths corresponding to the user cluster, we create a scatter plot (see Fig. 2) and measure the Spearmann correlation coefficient among the two variables [6], obtaining a negative correlation $\rho = -0.48$ with a two-sided p-value $p = 0.02$. Both the plot and the correlation coefficient thus appear to show that the amount of training data is negatively correlated with the perplexity of the model, supporting the intuition that the size of the training set is actually enhancing the model.

Another factor that might influence the performance in terms of next category prediction across the different user clusters is the fact that the average path length varies. Similarly to the previous analysis, we plot average path length and model perplexity (see Fig. 3) and measure the Spearmann correlation between the variables, obtaining a position correlation $\rho = 0.49$ with a two-sided p-value $p = 0.02$.

Gender	Lang	Paths	avg_l	max_l	ppl
M	All	18,718	4.23	50	75.478
F	All	8,741	4.16	38	73.024
All	En	8,955	3.96	35	71.343
All	Ar	439	4.15	28	100.772
All	Es	1,745	3.79	17	89.994
All	Ja	7,532	5.09	50	84.534
All	Nl	293	4.01	37	112.966
All	Pt	2,713	3.76	23	72.954
All	Th	795	4.19	26	90.343
All	Tr	6,142	3.85	31	66.838
F	En	2,636	3.94	28	74.999
F	Es	480	3.81	17	71.223
F	Ja	2,290	4.89	37	98.801
F	Pt	780	3.79	22	78.355
F	Th	234	4.05	15	83.601
F	Tr	1,863	3.80	18	82.078
M	En	5,771	3.95	35	69.481
M	Es	1,164	3.74	17	84.008
M	Ja	4,726	5.18	50	90.437
M	Pt	1,778	3.76	23	70.189
M	Th	524	4.30	26	94.132
M	Tr	3,886	3.89	31	69.056
All	All	29,465	4.20	50	75.271

Table 4: Personalized model vs global model.

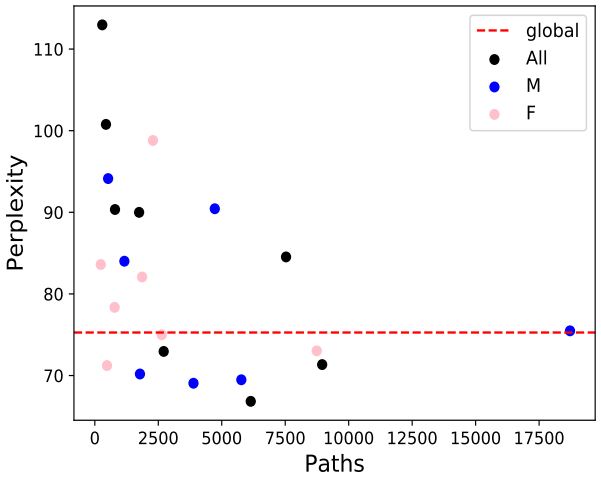

Figure 2: Number of paths in user cluster U_i and model perplexity. Blue circle correspond to "M", pink circles to "F" and black circles to "All". The horizontal dashed line correspond to the global model, including all users.

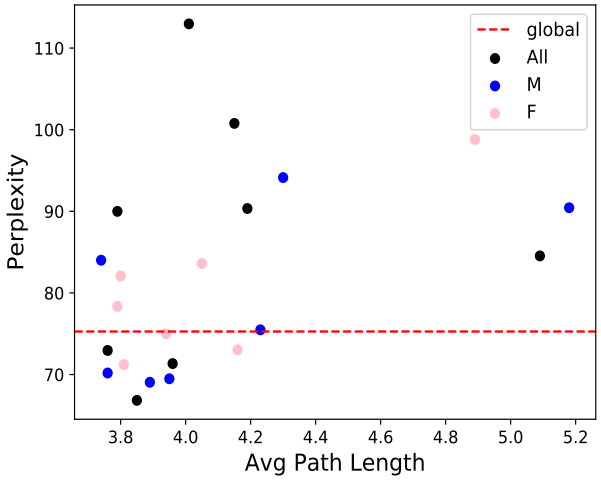

Figure 3: Average path length for user cluster U_i and model perplexity. Blue circle correspond to "M", pink circles to "F" and black circles to "All". The horizontal dashed line correspond to the global model, including all users.

6 CONCLUSIONS

In this paper, we propose a novel approach to recommend sequences of POI categories, as a first step to create a system that is able to automatically learn from data a personalized tourist path. The approach is based on a Recurrent Neural Network model, which shows to be able to model and predict effectively sequences of POI categories. We experiment different hyper parameters of the architecture of the network, showing the importance of a small learning rate and of stacking up multiple layers rather than increasing the number of neurons in the hidden layers. We also show that initializing the categories using an encoding based on node2vec improves the performance of the model with respect to the standard one-hot encoding, both in terms of model perplexity and of computing time. The analysis of the results of the model using different user clusters has a less definite interpretation, as we observe that in certain cases the performance increases and in other cases the performance decreases. We suggest that possible biasing factors are the size of the training set and the average path length, which is confirmed by a correlation analysis with the perplexity of the model. Further studies will extend the analyses to a larger dataset with a larger sample of user to rule out finite size effects on the performance of the clustered models and to include a larger temporal interval to exclude from the analysis possible seasonal effects.

Future work will also involve the integration of the next POI prediction into a real recommender of sequences of POIs for tourists. Given the next most likely POI category, a short list of sorted POIs belonging to that category will be retrieved from a knowledge base containing places and events. The short list will be defined according to a set of features modeling the user context (e.g. geographical position), inherent venue peculiarities (e.g. ratings, reviews) and user preferences.

ACKNOWLEDGMENTS

This work was partially supported by the innovation activity PasTime (17164) of EIT Digital.

REFERENCES

[1] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research* 3, Feb (2003), 1137–1155.

[2] Christopher M Bishop. 2006. Pattern recognition. *Machine Learning* 128 (2006), 1–58.

[3] Stanley F Chen and Joshua Goodman. 1996. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 310–318.

[4] Chen Cheng, Haiqin Yang, Michael R Lyu, and Irwin King. 2013. Where You Like to Go Next: Successive Point-of-Interest Recommendation.. In *IJCAI*, Vol. 13. 2605–2611.

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[6] Gregory W Corder and Dale I Foreman. 2014. *Nonparametric statistics: A step-by-step approach*. John Wiley & Sons.

[7] Munmun De Choudhury, Moran Feldman, Sihem Amer-Yahia, Nadav Golbandi, Ronny Lempel, and Cong Yu. 2010. Automatic construction of travel itineraries using social breadcrumbs. In *Proceedings of the 21st ACM conference on Hypertext and hypermedia*. ACM, 35–44.

[8] Shanshan Feng, Xutao Li, Yifeng Zeng, Gao Cong, Yeow Meng Chee, and Quan Yuan. 2015. Personalized Ranking Metric Embedding for Next New POI Recommendation.. In *IJCAI*. 2069–2075.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[10] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 6645–6649.

[11] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 855–864.

[12] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[13] Andrej Karpathy and Li Fei-Fei. 2015. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3128–3137.

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

[15] S Kotiloglu, T Lappas, K Pelechrinis, and PP Repoussis. 2017. Personalized multi-period tour recommendations. *Tourism Management* 62 (2017), 76–88.

[16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[17] Yanhua Li, Moritz Steiner, Limin Wang, Zhi-Li Zhang, and Jie Bao. 2013. Exploring venue popularity in foursquare. In *INFOCOM, 2013 Proceedings IEEE*. IEEE, 3357–3362.

[18] Bin Liu and Hui Xiong. 2013. Point-of-interest recommendation in location based social networks with topic and location awareness. In *Proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM, 396–404.

[19] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of Machine Learning Research* 9, Nov (2008), 2579–2605.

[20] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model.. In *Interspeech*, Vol. 2. 3.

[21] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[22] Anastasios Noulas, Salvatore Scellato, Cecilia Mascolo, and Massimiliano Pontil. 2011. Exploiting Semantic Annotations for Clustering Geographic Areas and Users in Location-based Social Networks. *The Social Mobile Web* 11, 2 (2011).

[23] Daniel Preoţiuc-Pietro, Justin Cranshaw, and Tae Yano. 2013. Exploring venue-based city-to-city similarity measures. In *Proceedings of the 2nd ACM SIGKDD International Workshop on Urban Computing*. ACM, 16.

[24] Giuseppe Rizzo, Rosa Meo, Ruggero G Pensa, Giacomo Falcone, and Raphaël Troncy. 2017. Shaping City Neighborhoods Leveraging Crowd Sensors. *Information Systems* 64 (2017), 368–378.

[25] G. Rizzo, R. Troncy, O. Corcho, A. Jameson, J. Plu, J.C. Ballesteros Hermida, A. Assaf, C. Barbu, A. Spirescu, K. Kuhn, I. Celino, R. Agarwal, C.K. Nguyen, A. Pathak, C. Scanu, M. Valla, T. Haaker, E.S. Verga, M. Rossi, and J.L. Redondo Garcia. 2015. 3cixty@Expo Milano 2015: Enabling Visitors to Explore a Smart City. In *14^{th} International Semantic Web Conference (ISWC), Semantic Web Challenge*.

[26] Jrgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61 (2015), 85–117.

[27] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[28] Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 1017–1024.

[29] Duyu Tang, Bing Qin, and Ting Liu. 2015. Document Modeling with Gated Recurrent Neural Network for Sentiment Classification.. In *EMNLP*. 1422–1432.

[30] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. 2011. The orienteering problem: A survey. *European Journal of Operational Research* 209, 1 (2011), 1–10.

[31] Jihang Ye, Zhe Zhu, and Hong Cheng. 2013. What's your next move: User activity prediction in location-based social networks. In *Proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM, 171–179.

[32] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. 2008. Understanding mobility based on GPS data. In *Proceedings of the 10th international conference on Ubiquitous computing*. ACM, 312–321.