Recommending Structured Objects: Paths and Sets

Dawei Chen

A thesis submitted for the degree of
Doctor of Philosophy of
The Australian National University

August 2019

Declaration

The majority of the work in this thesis has been published in conference proceedings and preprints, I list them below.

- **Dawei Chen**, Cheng Soon Ong, Lexing Xie.
  Learning points and routes to recommend trajectories.
  In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 2227–2232. 2016.

- **Dawei Chen**, Dongwoo Kim, Lexing Xie, Minjeong Shin, Aditya Krishna Menon, Cheng Soon Ong, Iman Avazpour, John Grundy.
  PathRec: Visual analysis of travel route recommendations.
  In *Proceedings of the ACM Recommender Systems Conference*, pages 364–365. 2017.

- Aditya Krishna Menon, **Dawei Chen**, Lexing Xie, Cheng Soon Ong.
  Revisiting revisits in trajectory recommendation.
  In *Proceedings of the International Workshop on Recommender Systems for Citizens*, pages 2:1–2:6. 2017.

- **Dawei Chen**, Lexing Xie, Aditya Krishna Menon, Cheng Soon Ong.
  Structured recommendation.
  CoRR, abs/1706.09067. 2017. https://arxiv.org/abs/1706.09067

- **Dawei Chen**, Cheng Soon Ong, Aditya Krishna Menon.
  Cold-start playlist recommendation with multitask learning.
  CoRR, abs/1901.06125. 2019. https://arxiv.org/abs/1901.06125

Except where otherwise indicated, this thesis is my own original work.

Dawei Chen
12 August 2019

Primary Supervisor

Cheng Soon Ong
Principal Research Scientist, Data61, CSIRO
Adjunct Associate Professor, The Australian National University
Canberra, ACT, Australia

Associate Supervisors

Lexing Xie
Professor, The Australian National University
Canberra, ACT, Australia

Aditya Krishna Menon
Honorary Senior Lecturer, The Australian National University
Senior Research Scientist, Google
New York, NY, USA

To my family.

Acknowledgement

First, I would like to express my deepest gratitude to my primary supervisor, Dr. Cheng Soon Ong, for his inspiration and guidance during my doctoral study. I greatly appreciate his tireless efforts for helping me formulate my research problems as well as guiding me towards making good decisions among the exponential number of possible design choices. This thesis would not be possible if not for his dedication. I would like to sincerely thank my associate supervisor, Dr. Lexing Xie, for her support of my PhD application and her invaluable guidance during the past few years. I greatly appreciate her help in presenting my work in CIKM when my visa was granted too late to attend the conference. My sincere thanks also goes to my associate supervisor, Dr. Aditya Krishna Menon, for contributing a large number of hours to help with my research, including but not limited to deriving equations, editing paper drafts and challenging me to prove theorems. His style of working from the most basic principles has always been inspiring to me.

I would also like to thank all members in the ANU Computational Media Lab for organising various fun group activities and social events. I am indebted to the Australian National University, NICTA and CSIRO Data61 for providing financial and technical support for my research. Last but not the least, I want to thank my family for their encouragement and constant support.

Abstract

Recommender systems have been widely adopted in industry to help people find the most appropriate items to purchase or consume from the increasingly large collection of available resources (e.g., books, songs and movies). Conventional recommendation techniques follow the approach of "ranking all possible options and pick the top", which can work effectively for single item recommendation but fall short when the item in question has internal structures. For example, a travel trajectory with a sequence of points-of-interest or a music playlist with a set of songs. Such structured objects pose critical challenges to recommender systems due to the intractability of ranking all possible candidates.

This thesis study the problem of recommending structured objects, in particular, the recommendation of *path* (a sequence of unique elements) and *set* (a collection of distinct elements). We study the problem of recommending travel trajectories in a city, which is a typical instance of path recommendation. We propose methods that combine learning to rank and route planning techniques for efficient trajectory recommendation. Another contribution of this thesis is to develop the structured recommendation approach for path recommendation by substantially modifying the loss function, the learning and inference procedures of structured support vector machines. A novel application of path decoding techniques helps us achieve efficient learning and recommendation. Additionally, we investigate the problem of recommending a set of songs to form a playlist as an example of the set recommendation problem. We propose to jointly learn user representations by employing the multi-task learning paradigm, and a key result of equivalence between bipartite ranking and binary classification enables efficient learning of our set recommendation method. Extensive evaluations on real world datasets demonstrate the effectiveness of our proposed approaches for path and set recommendation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The ever increasing number of available items from content providers (e.g., news articles and blog posts from online publishing platforms, books from Amazon book store, music from Spotify and Apple Music, and movies from Netflix library) pose critical challenges for both users and content providers. This information overload problem makes it particularly difficult for users to find items they want, and further, it undermines the effort to facilitate users getting the most appropriate items that content providers have tirelessly been striving to achieve. Such challenges motivate the development of *recommender system*, a type of information system that suggests items which (hopefully) match users' preferences. Good recommendations enhance user satisfaction (Bennett et al., 2007; Koren et al., 2009; Aggarwal, 2016), which helps content providers to be more successful, and therefore enables the further expansion of available items to attract even more users. This in turn makes it more challenging to provide good recommendations that meet user needs.

1.1 Motivation

Classic recommender systems (e.g., those used for book or movie recommendation) work by ranking items for a particular user. One of the most widely used techniques for recommender systems of this type is the collaborative filtering approach (Goldberg et al., 1992), in particular, latent factor models such as the matrix factorisation family of techniques (Koren et al., 2009). These methods work by explicitly matching user preferences with item properties, both of which are latent factors learned from historical interaction records between users and items (e.g., user's ratings of items, histories of purchases or browsing). The advantage of this technique is the ability to automatically learn informative representations of users and items from historical data, and the capability to scale to large systems with millions of users and items, and some variants can even capture the temporal dynamics of user preferences (Koren et al., 2009; Koren, 2009; Rendle et al., 2010; Xiong et al., 2010).

However, there are at least two limitations of such approaches. First, it suffers from addressing users or items without historical records, since it is unlikely to learn any meaningful representations for these new users and items; this is known as the *cold start* problem (Schein et al., 2002; Koren et al., 2009). Further, this approach makes a recommendation for a user by scoring each item for that user and suggests the top scored ones, which falls short when dealing with structured objects (i.e., object that is a cohesive composition of interrelated and interdependent elements) effectively, since the huge number of possible combinations of elements is unlikely to be scored efficiently if one adopts a naïve approach (Taskar, 2004; Joachims et al., 2009b).

On the other hand, the task of recommending structured objects is prevalent in real-world applications, for example, recommending a trajectory of points-of-interest (POIs) in a city to a visitor (Lu et al., 2010, 2012; Lim et al., 2015; Chen et al., 2016; He et al., 2018), suggesting a chemical compound (Dehaspe et al., 1998; Agrafiotis et al., 2007), or a playlist of songs (McFee and Lanckriet, 2011; Chen et al., 2012; Choi et al., 2016; Ben-Elazar et al., 2017). A trajectory is a sequence of distinct POIs, which is a *path*; and a music playlist involves a *set* of unique songs. Such tasks of recommending structured objects (e.g., *paths* and *sets*), which are both practically important and computationally hard, motivate the work in this thesis.

1.2 Contributions

In this thesis, we develop techniques that can efficiently recommend objects with different types of structures, in particular, *paths* and *sets*. Machine learning techniques including learning to rank and structured prediction, as well as route planning techniques are employed to achieve efficient path recommendation. We empirically evaluate these techniques on the task of recommending travel trajectories. In addition, we develop a technique for set recommendation by leveraging bipartite ranking and multi-task learning, and evaluate it on the task of recommending a set of songs from a music library. The major contributions of this thesis are summarised as follows:

1. **Efficient path recommendation via point ranking and route planning.** We study the problem of recommending paths, which is to recommend a sequence of elements in a certain order without repetition. Specifically, we investigate the problem of recommending travel trajectories, which is an instance of path recommendation, to visitors in a city. We propose a new formulation of this problem by combining two widely studied problems, i.e., learning to rank (Burges et al., 2005; Cao et al., 2007; Liu, 2009) and the travelling salesman problem (TSP) (Miller et al., 1960; Applegate et al., 2011). We develop an approach that

can efficiently recommend travel trajectories by employing the RankSVM (Lee and Lin, 2014; Kuo et al., 2014) and route planning techniques from the research of the TSP or the orienteering problem (Miller et al., 1960; Golden et al., 1987; Applegate et al., 2011). This work is published in Chen et al. (2016).

2. **Structured recommendation for paths based on structured prediction.** We study the problem of recommending paths from a structured prediction perspective (Tsochantaridis et al., 2004; Taskar et al., 2005; BakIr et al., 2007; Joachims et al., 2009b), and propose a new structured recommendation approach based on the structured support vector machines (SSVMs) framework. We analyse the fundamental challenges of travel trajectory recommendation, which are shared among many path recommendation tasks, and show how the structured recommendation approach can address these challenges by systematically incorporating point preferences and transition patterns, as well as a novel application of the list variant of the Viterbi decoding algorithm for hidden Markov models (Soong and Huang, 1991; Seshadri and Sundberg, 1994; Nill and Sundberg, 1995; Nilsson and Goldberger, 2001), or the integer linear programming formulation of the *s-t path* TSP (Hoogeveen, 1991; An et al., 2015). This work is presented in Chen et al. (2017a,b) and Menon et al. (2017).

3. **Efficient set recommendation via bipartite ranking and multi-task learning.** We study the problem of recommending sets, which is to recommend a collection of distinct elements where no particular order of the elements are of interest. We propose a new approach for set recommendation via bipartite ranking (Agarwal and Niyogi, 2005; Ertekin and Rudin, 2011; Menon and Williamson, 2016). Specifically, we investigate the problem of recommending a set of songs from a music library to form a new playlist or extend a user's existing playlist in cold-start scenarios. A bipartite ranking loss is adopted to encourage songs in a playlist to be ranked higher than those not in it, and the multi-task learning paradigm is employed to jointly learn user representations that facilitate cold-start recommendation. Lastly, we achieve efficient learning of the set recommendation approach by leveraging a key equivalence between bipartite ranking and binary classification (Ertekin and Rudin, 2011; Menon and Williamson, 2016). This work is presented in Chen et al. (2019).

In addition to the above contributions, in Appendix B, we generalise the results of the equivalence between classification and ranking presented in Ertekin and Rudin (2011) and show that, under some conditions, the equivalence also holds for a parametric family of binary classification losses and a parametric family of bipartite ranking losses. This result is presented for the first time in this thesis.

1.3 Thesis outline

The rest of this thesis is outlined as follows. In Chapter 2, we first define the problem of recommending structured objects. In particular, the problems of interest in this thesis, i.e., the path and set recommendation problems are formulated. We then present classic techniques for recommender systems, and review essential techniques that help us achieve efficient recommendation of paths and sets, including structured prediction and path decoding techniques in Markov chains for path recommendation; and loss functions of bipartite ranking and binary classification as well as the multi-task learning paradigm that enables effective set recommendation.

In Chapter 3, we study the problem of recommending path by investigating one particular example – travel trajectory recommendation, which is to suggest a sequence of points-of-interest (POIs) without repetition. We first show that both the ranking of POIs and the transition patterns between POIs are helpful in recommending travel trajectories, and then present a method that combines POI ranking and route planning to achieve effective recommendation of travel trajectories.

We continue the study of the path recommendation problem in Chapter 4. Here we investigate the travel trajectory recommendation problem from a structured prediction viewpoint. We propose the structured recommendation approach for path recommendation by modifying the loss function of the SSVMs to aggregate multiple ground truths for a query. Further, we show how a novel application of the list variant of the classic Viterbi algorithm in both training and inference procedures of the SSVMs can help us overcome the challenges of trajectory recommendation.

In Chapter 5, we study the problem of recommending sets. We investigate an instance of this problem, which is to recommend a set of songs from a music library to form a new playlist or extend an existing playlist in cold-start scenarios. We employ a bipartite ranking loss to rank songs in a playlist above those that are not in it, and adopt the multi-task learning paradigm to learn the representations of all users jointly. A key equivalence between bipartite ranking and binary classification enables the efficient learning of our set recommendation approach from historical playlists.

Lastly, in Chapter 6, we summarise the work in this thesis, i.e., the path and set recommendation techniques and their applications in recommending travel trajectories and music playlists. We then present the limitations of our approaches to recommending structured objects as well as future research directions.

Chapter 2

Background

We review important problems and techniques that serve as the foundation of our work in this thesis. First, we define the problem of recommending structured objects, in particular, we formulate the path and set recommendation problems (Section 2.1). We then review classic techniques for recommender systems (Section 2.2). In Sections 2.3 and 2.4, we present structured prediction and path decoding techniques that enables efficient path recommendation, followed by techniques for set recommendation including bipartite ranking and binary classification (Section 2.5), as well as the multi-task learning paradigm (Section 2.6). Table 2.1 shows the corresponding problems and chapters of this thesis that make use of the techniques we reviewed.

2.1 The problem of recommending structured objects

The problem of recommendation is to suggest items (e.g., books, movies and news) that a user may like. In this thesis, we consider the case when the item in question is an object with internal structures (i.e., object that is a cohesive composition of interrelated and interdependent elements), such as a set or a tree.

Let $\mathcal{Y}$ be the space of all structured objects, given a set of N *queries* that describe the specifications of desired recommendations, and the historical records (e.g., items purchased or consumed by users) of the corresponding queries

$$\mathcal{S} = \{(q^{(i)}, \{\mathbf{y}^{(ij)}\}_{j=1}^{N_i})\}_{i=1}^{N}, \quad \mathbf{y}^{(ij)} \in \mathcal{Y},\ i \in \{1, \dots, N\},\ j \in \{1, \dots, N_i\}, \tag{2.1}$$

where the i-th query $q^{(i)}$ is associated with $N_i \in \mathbb{Z}^+$ objects. The problem of *recommending structured objects* is to make recommendations for a new query q not seen in $\mathcal{S}$, in other words, we learn a function f from $\mathcal{S}$:

$$f : q \to \{\mathbf{y}^{(k)}\}_{k=1}^{K}, \quad \mathbf{y}^{(k)} \in \mathcal{Y}, \tag{2.2}$$

where $K \in \mathbb{Z}^+$ is the number of structured objects we shall recommend for query q.

Table 2.1: Techniques for recommending structured objects.

Technique	Recommendation problem
Structured prediction (§2.3)	Path recommendation (Chapter 4)
Path decoding in Markov chains (§2.4)	Path recommendation (Chapters 3 and 4)
Binary classification and bipartite ranking (§2.5)	Set recommendation (Chapter 5)
Multi-task learning (§2.6)	Set recommendation (Chapter 5)

Compared to standard supervised learning, there are multiple ground truth labels (i.e., recommended objects) in the training set (2.1). Further, in contrast to a fixed set of individual items, the huge number of all possible structured labels (i.e., $|\mathcal{Y}|$) poses a critical challenge to efficient recommendation. Lastly, taking into account constraints inherent to particular types of structures, for example, a *set* should not have duplicate elements, presents another fundamental challenge.

This thesis aims to develop methods that can address these challenges in recommending structured objects, particularly for two types of structures: *path* and *set*. We remark that the recommendation of other types of structures (e.g., tree or graph) are important research topics on their own, and we leave them as future work.

2.1.1 Path recommendation

We define a *path* as a sequence of elements where no element in the sequence appears more than once. The path recommendation problem is a particular instance of the problem of recommending structured objects, where the objects in question are paths. Specifically, given a set of elements $\mathcal{P}$, a structured object in both (2.1) and (2.2) is a *path* with elements from $\mathcal{P}$, i.e.,

$$\mathbf{y} \in \mathcal{P}^{|\mathbf{y}|} \text{ and } y_l \neq y_{l'}, \quad l, l' \in \{1, \ldots, |\mathbf{y}|\}, l \neq l',$$

where $|\mathbf{y}|$ denotes the number of elements in path $\mathbf{y}$.

A particular example of the path recommendation problem is travel trajectory (or tour) recommendation, where we suggest tours (i.e., sequences of distinct POIs) to visitors that satisfy certain properties (e.g., starting and ending at specific locations, visiting a particular number of POIs) which shall be specified in a trajectory query. In Chapter 3, we discuss a path recommendation approach for suggesting travel trajectories, where we leverage POI rankings and transition preferences to recommend tours. We further develop a structured recommendation approach for recommending paths in Chapter 4, which is based on structured prediction and path decoding techniques that will be reviewed in Section 2.3 and Section 2.4, respectively.

2.1.2 Set recommendation

A *set* is a collection of distinct elements, i.e., no element in the collection appears more than once, and there is also no particular order between elements. The set recommendation problem is another example of the problem of recommending structured objects, where the objects in question are sets. In particular, given a set of elements $\mathcal{C}$, the structured object space $\mathcal{Y}$ in (2.1) and (2.2) is the power set of $\mathcal{C}$.

In Chapter 5, we study an instance of the set recommendation problem – music playlist recommendation,[1] where we recommend a set of songs from a music library to form a new playlist or extend a user's existing playlist, with respect to certain properties specified in a query. We discuss a set recommendation method for music playlist recommendation in three cold-start settings. Our approach employs the multi-task learning paradigm (Section 2.6) to jointly learn user representations, and efficient learning of our set recommendation method is achieved by leveraging a key equivalence between bipartite ranking and binary classification (Section 2.5).

2.2 Techniques for recommender systems

In this section, we review classic techniques underlying typical recommender systems (also known as recommendation systems). First, we present the general recommendation strategies, then we review a family of techniques widely employed in practice.

2.2.1 Recommendation strategies

To suggest items to a user, a recommender system generally ranks items by matching their properties with the preferences of the user, and top-ranked items are then recommended. User preferences and item properties are often encoded using a vector of real numbers, which are known as the representation or features of a user or an item. Features can be extracted either manually by human experts (e.g., the Music Genome Project that powers the Pandora Radio (John, 2006)) or learned automatically by computer algorithms. The former is known as the *content filtering* approach, and the latter is known as the *collaborative filtering* approach (Goldberg et al., 1992), including, for example, the neighbourhood method and latent factor models.

Content filtering approaches rely on human experts to extract (user or item) features, with the advantage of leveraging knowledge accumulated in a certain

[1] Music playlist can also be formulated as a path recommendation problem which emphasises the listening order of songs in a playlist, however, whether the order of songs in a playlist is critical is still not clear from previous research; further, we observe that it is not uncommon to listen a playlist in shuffle mode, where a random listening order is generated each time, we therefore focus on suggesting a set of songs for music playlist recommendation.

domain; however, different experts may devise different values for a certain feature of the same user or item due to the variety in backgrounds and experiences of human experts. In addition, this approach cannot be applied on a large scale simply because the number of human experts in a certain domain is limited.

On the other hand, collaborative filtering approaches leverage historical interaction data to make recommendations. For example, the neighbourhood method makes recommendations by using information from similar users and similar items:[2] for a particular user u, it recommends items purchased by users similar to u, or suggests items similar to what u has purchased. Latent factor models, on the other hand, learn user and item features (i.e., latent factors) from data of interactions between users and items. For example, learning a low-rank matrix that approximates to the matrix of user-item interactions (e.g., ratings), which is generally very sparse, by exploiting the redundancies in interaction data (Aggarwal and Parthasarathy, 2001).

A major drawback of collaborative filter approaches is the inability to address the *cold-start* scenarios, where there is no historical data for either users or items, which occurs when new users or new items are added to an existing system (Koren et al., 2009; Aggarwal, 2016). In this case, the content filtering approach can generally work more effectively as it can leverage external information of the new users or items provided by human experts.

2.2.2 Matrix factorisation techniques

One of the most successful latent factor models is the matrix factorisation (MF) family of techniques, these learn the representation or features of users and items by factorising a matrix with historical records, e.g., ratings of books or movies.

Given a set of N users and a set of M items, Matrix Factorisation approximates the user-item rating matrix $R \in \mathbb{R}^{N \times M}$ using the product of two low rank matrices, i.e., $\hat{R} = UV^T$, as illustrated in Figure 2.1, where $U \in R^{N \times D}$ and $V \in \mathbb{R}^{M \times D}$ are low rank matrices, with row vectors $\mathbf{u}_i$, $i \in \{1, \ldots, N\}$ and $\mathbf{v}_j$, $j \in \{1, \ldots, M\}$ representing the D dimensional latent features of user i and item j respectively.

Due to the fact that the rating matrix R is generally highly sparse (i.e., most of the entries in R are not observed), classic matrix decomposition methods such as the Singular Value Decomposition (SVD) are not applicable.[3] In addition, data imputation could be very costly given the large number of unobserved ratings in

[2] A user u is regarded to be similar to another user u' if both u and u' have purchased (or have given similar ratings to) the same set of items, in other words, u and u' are like-minded users; similarly, an item v is regarded to be similar to another item v' if both v and v' have been purchased by (or have received similar ratings from) the same user.

[3] SVD is only defined for matrices where every entry is observed.

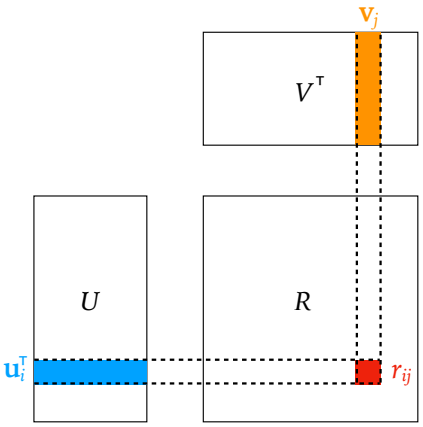


Figure 2.1: Illustration of Matrix Factorisation. U is a matrix of latent factors of users, V is a matrix of latent factors of items, and R is the observed (sparse) rating matrix. Matrix Factorisation aims to approximate the observed ratings using a function of the corresponding latent factors of users and items, e.g., $r_{ij} \approx \mathbf{u}_i^\top \mathbf{v}_j$.

practical datasets, and inaccurate imputation could considerably distort the data. As a result, many works propose to model the observed ratings directly by minimising a regularised squared error (Paterek, 2007; Takács et al., 2007; Koren et al., 2009),

$$\min_{U,V} \sum_{i=1}^{N}\sum_{j=1}^{M} \delta_{ij}\left(r_{ij} - \mathbf{u}_i^\top \mathbf{v}_j\right)^2 + C\left(\sum_{i=1}^{N}\|\mathbf{u}_i\|^2 + \sum_{j=1}^{M}\|\mathbf{v}_j\|^2\right), \tag{2.3}$$

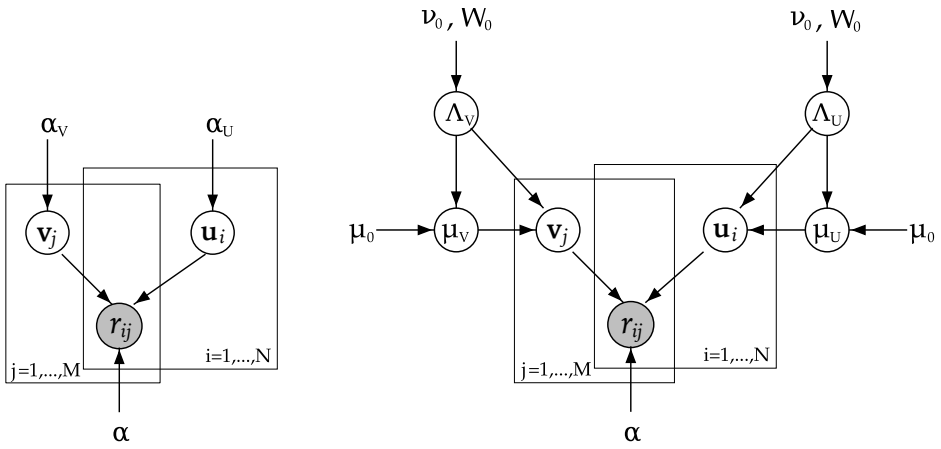
where δ_{ij} indicates whether user i has rated item j (i.e., $\delta_{ij} = 1$ if entry r_{ij} is observed and $\delta_{ij} = 0$ otherwise), and $C \in \mathbb{R}^+$ is a regularisation constant.

The objective in Equation (2.3) is non-convex, which makes it hard to find an optimal solution of the problem. However, optimisation methods such as stochastic gradient descent (SGD) and coordinate descent (e.g., alternating least squares (ALS)) have been shown to work effectively in practice (Hu et al., 2008; Koren et al., 2009).

Probabilistic Matrix Factorisation (PMF) (Salakhutdinov and Mnih, 2008b) provides a probabilistic explanation to the regularisation in Equation (2.3). PMF models the user-item rating R_{ij} using a Gaussian random variable, with mean $\mu_{ij} = \mathbf{u}_i^\top \mathbf{v}_j$ and precision α. Assuming zero mean spherical Gaussian priors over the latent feature vectors of both users and items, it has been shown that maximising the log-posterior over U and V is equivalent to

$$\min_{U,V} \sum_{i=1}^{N}\sum_{j=1}^{M} \delta_{ij}\left(r_{ij} - \mathbf{u}_i^\top \mathbf{v}_j\right)^2 + \frac{\alpha_U}{\alpha}\sum_{i=1}^{N}\|\mathbf{u}_i\|^2 + \frac{\alpha_V}{\alpha}\sum_{j=1}^{M}\|\mathbf{v}_j\|^2, \tag{2.4}$$

where α_U and α_V are hyper-parameters. Figure 2.2(a) shows the graphical model of the Probabilistic Matrix Factorisation.



(a) PMF (b) BPMF

Figure 2.2: Graphical models of (a) Probabilistic Matrix Factorisation (PMF) and (b) Bayesian Probabilistic Matrix Factorisation (BPMF). Shaded nodes: observed variables; unshaded nodes: latent variables (Salakhutdinov and Mnih, 2008a).

One approach to tune the hyper-parameters α, α_U and α_V in Equation (2.4) is to search over a set of carefully selected values based on the model performance on a validation set, however, this is computationally expensive as a large number of models should to be trained. Alternatively, one can take a Bayesian approach, i.e., introducing priors for hyper-parameters. Salakhutdinov and Mnih (2008a) proposed the Bayesian Probabilistic Matrix Factorisation (BPMF) which assumes more general Gaussian priors over the latent feature vectors of users and items instead of zero mean Gaussian priors as did in the PMF approach,

$$P(U \mid \mu_U, \Lambda_U) = \prod_{i=1}^{N} \mathcal{N}(\mathbf{u}_i \mid \mu_U, \Lambda_U^{-1}),$$
$$P(V \mid \mu_V, \Lambda_V) = \prod_{j=1}^{M} \mathcal{N}(\mathbf{v}_j \mid \mu_V, \Lambda_V^{-1}).$$

It further assumes Gaussian-Wishart priors over hyper-parameters $\Theta_U = \{\mu_U, \Lambda_U\}$ and $\Theta_V = \{\mu_V, \Lambda_V\}$,

$$P(\Theta_U \mid \Theta_0) = \mathcal{N}(\mu_U \mid \mu_0, (\beta_0 \Lambda_U)^{-1}) \mathcal{W}(\Lambda_U \mid W_0, \nu_0),$$
$$P(\Theta_V \mid \Theta_0) = \mathcal{N}(\mu_V \mid \mu_0, (\beta_0 \Lambda_V)^{-1}) \mathcal{W}(\Lambda_V \mid W_0, \nu_0),$$

where $\Theta_0 = \{\mu_0, W_0, \nu_0\}$, and $\mathcal{W}$ is the Wishart distribution with degrees of freedom ν_0 and scale matrix $W_0 \in \mathbb{R}^{D \times D}$. Figure 2.2(b) shows the graphical model of the Bayesian Probabilistic Matrix Factorisation.

To obtain the posterior distribution, all model parameters and hyper-parameters need to be integrated out, which is analytically intractable. Salakhutdinov and Mnih (2008a) proposed to approximate the posterior distribution using a Markov chain Monte Carlo (MCMC) method, in particular, a Gibbs sampling algorithm that samples from conditional distributions over model parameters and hyper-parameters.

Matrix factorisation techniques have also been generalised to deal with implicit feedback, e.g., history of purchasing, ad clicking and browsing. Hu et al. (2008) proposed to use a matrix with binary values to represent implicit feedback, and minimised an objective similar to that in Equation (2.3) but with a weighted square loss. Rendle et al. (2009) observed that only positive and unlabelled data are available for implicit feedback, they proposed to dealt with it from a ranking perspective: For a particular user, the approach ranked observed items (e.g., items purchased or viewed by that user) higher than all other items. Other variants of MF that can deal with specific scenarios have been proposed, such as those that can incorporate additional sources of information (Paterek, 2007; Koren, 2008; Koren et al., 2009), and variants that can capture temporal effects (Koren, 2009; Koren et al., 2009; Xiong et al., 2010).

2.3 Structured prediction

Structured prediction is the task of predicting interdependent output variables for a given input (BakIr et al., 2007), typical applications including multi-class and multi-label classification, image segmentation and machine translation. The primary challenge in structured prediction is how to make efficient inference in the exponentially-sized output space. In this section, we review an important technique for structured prediction, the structured support vector machines (SSVMs), which generalises the support vector machines (SVMs) to the structured output setting. Another important technique for structured prediction is the conditional random fields (CRFs), Pletscher et al. (2010) proposed a framework that unifies CRFs and SSVMs, we therefore focus on reviewing the SSVMs in this section. In Chapter 4, we formulate the problem of recommending paths in the context of travel trajectories as a structured prediction problem, and adapt both the training and prediction procedures of the SSVMs to deal with challenges in recommending paths.

2.3.1 Structured support vector machines

Consider the task of predicting some structured label, e.g., a tree, let $\mathcal{X}$ be the instance space, and $\mathcal{Y}$ be the space of all possible structured labels, e.g., all possible trees. The scoring function $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ (parameterised by $\mathbf{w}$) measures the affinity between

a particular example $\mathbf{x} \in \mathcal{X}$ and a specific structured label $\mathbf{y} \in \mathcal{Y}$. The structured support vector machines makes a prediction given example $\mathbf{x}$ by

$$\mathbf{y}^* = \operatorname*{argmax}_{\bar{\mathbf{y}} \in \mathcal{Y}} f(\mathbf{x}^{(i)}, \bar{\mathbf{y}}), \tag{2.5}$$

which is known as the *inference* or prediction procedure of the SSVMs.

To learn the parameters $\mathbf{w}$, the SSVMs minimises the following (ℓ_2 regularised) empirical risk on dataset $\mathcal{S} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$:

$$\min_{\mathbf{w}} \ \frac{1}{2}\mathbf{w}^\top \mathbf{w} + \frac{C}{N}\sum_{i=1}^{N} \ell(\mathbf{y}^{(i)}, f(\mathbf{x}^{(i)}, \cdot)), \tag{2.6}$$

where $C \in \mathbb{R}^+$ is a regularisation constant and $\ell(\mathbf{y}^{(i)}, f(\mathbf{x}^{(i)}, \cdot))$ is the structured hinge loss for the i-th labelled example

$$\ell(\mathbf{y}^{(i)}, f(\mathbf{x}^{(i)}, \cdot)) = \max\left(0, \ \max_{\bar{\mathbf{y}} \in \mathcal{Y}} \left\{\Delta(\mathbf{y}^{(i)}, \bar{\mathbf{y}}) - (f(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) - f(\mathbf{x}^{(i)}, \bar{\mathbf{y}}))\right\}\right), \tag{2.7}$$

here $\Delta(\mathbf{y}^{(i)}, \bar{\mathbf{y}})$ measures the discrepancy between two structured labels $\mathbf{y}^{(i)}$ and $\bar{\mathbf{y}}$.

If we employ a slack variable $\xi^{(i)}$ to upper bound the structured hinge loss for the i-th example, we can transform the unconstrained optimisation problem (2.6) into a constrained optimisation problem:

$$\begin{aligned} \min_{\mathbf{w}, \boldsymbol{\xi}} \ & \frac{1}{2}\mathbf{w}^\top \mathbf{w} + \frac{C}{N}\sum_{i=1}^{N} \xi^{(i)} \\ s.t. \ & \boldsymbol{\xi} \geq \mathbf{0}, \\ & \xi^{(i)} \geq \Delta(\mathbf{y}^{(i)}, \bar{\mathbf{y}}) - \left(f(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) - f(\mathbf{x}^{(i)}, \bar{\mathbf{y}})\right), \ \forall \bar{\mathbf{y}} \in \mathcal{Y}. \end{aligned} \tag{2.8}$$

This formulation is known as the "n-slack" SSVMs due to the number of slack variables in (2.8) equals the numbers of examples N (Joachims et al., 2009a).

We can rearrange the second constraint in problem (2.8) as

$$\xi^{(i)} + f(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \geq \Delta(\mathbf{y}^{(i)}, \bar{\mathbf{y}}) + f(\mathbf{x}^{(i)}, \bar{\mathbf{y}}), \ \forall \bar{\mathbf{y}} \in \mathcal{Y},$$

or equivalently,

$$\xi^{(i)} + f(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \geq \max_{\bar{\mathbf{y}} \in \mathcal{Y}} \left\{\Delta(\mathbf{y}^{(i)}, \bar{\mathbf{y}}) + f(\mathbf{x}^{(i)}, \bar{\mathbf{y}})\right\}. \tag{2.9}$$

The right-hand-side (RHS) of Equation (2.9) is known as the *loss-augmented inference*, which is closely related to the *inference* or prediction procedure of the SSVMs (Equa-

tion 2.5). In practice, if $\Delta(\mathbf{y}^{(i)}, \bar{\mathbf{y}})$ can be decomposed into discrepancies of individual elements in $\mathbf{y}^{(i)}$ and $\bar{\mathbf{y}}$, then the loss-augmented inference could be optimised using techniques similar to that for the inference procedure (2.5).

Joachims et al. (2009a) observed that one can sum up the structured hinge losses (Eq. 2.7) for all examples, which results in the following empirical risk on dataset $\mathcal{S}$:

$$R(f, \mathcal{S}) = \max\left(0, \max_{(\bar{\mathbf{y}}^{(1)}, \ldots, \bar{\mathbf{y}}^{(N)}) \in \mathcal{Y}^N} \left\{ \frac{1}{N} \sum_{i=1}^{N} \left(\Delta(\mathbf{y}^{(i)}, \bar{\mathbf{y}}^{(i)}) - \left(f(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) - f(\mathbf{x}^{(i)}, \bar{\mathbf{y}}^{(i)}) \right) \right) \right\} \right). \tag{2.10}$$

Similar to the "n-slack" SSVM (2.8), we can transform the (ℓ_2 regularised) empirical risk minimisation into a constrained optimisation problem, however, here only *one* slack variable is needed to upper bound the empirical risk $R(f, \mathcal{S})$ defined in (2.10),

$$\begin{aligned} \min_{\mathbf{w}, \xi} \quad & \frac{1}{2}\mathbf{w}^\top \mathbf{w} + C\xi \\ s.t. \quad & \xi \geq 0, \\ & \xi \geq \frac{1}{N} \sum_{i=1}^{N} \left(\Delta(\mathbf{y}^{(i)}, \bar{\mathbf{y}}^{(i)}) - (f(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) - f(\mathbf{x}^{(i)}, \bar{\mathbf{y}}^{(i)})) \right), \ \forall (\bar{\mathbf{y}}^{(1)}, \ldots, \bar{\mathbf{y}}^{(N)}) \in \mathcal{Y}^N. \end{aligned} \tag{2.11}$$

This formulation is known as the "1-slack" SSVMs (Joachims et al., 2009a).

If we rearrange the second constraint in (2.11) as

$$\xi + \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \geq \frac{1}{N} \sum_{i=1}^{N} \left(\Delta(\mathbf{y}^{(i)}, \bar{\mathbf{y}}^{(i)}) + f(\mathbf{x}^{(i)}, \bar{\mathbf{y}}^{(i)}) \right), \ \forall (\bar{\mathbf{y}}^{(1)}, \ldots, \bar{\mathbf{y}}^{(N)}) \in \mathcal{Y}^N$$

or equivalently

$$\xi + \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \geq \max_{(\bar{\mathbf{y}}^{(1)}, \ldots, \bar{\mathbf{y}}^{(N)}) \in \mathcal{Y}^N} \frac{1}{N} \sum_{i=1}^{N} \left(\Delta(\mathbf{y}^{(i)}, \bar{\mathbf{y}}^{(i)}) + f(\mathbf{x}^{(i)}, \bar{\mathbf{y}}^{(i)}) \right). \tag{2.12}$$

The RHS of Equation (2.12) is known as the *loss-augmented inference* in the "1-slack" formulation of the SSVMs.

Both the "n-slack" (2.8) and "1-slack" (2.11) formulations of the SSVMs result in constrained optimisation problems. If the affinity function $f(\mathbf{x}, \mathbf{y})$ takes the form of a linear function, then both formulations are quadratic programs (QPs). However, naively feeding these problems into an off-the-shelf QP solver is unlikely to work due to the huge number of constraints in (2.8) and (2.11). In particular, in the "n-slack" SSVMs, there is one constraint for every structured label in the set of labels $\mathcal{Y} \setminus \mathbf{y}^{(i)}$ for the i-th example, and the "1-slack" formulation further increases the number of constraints exponentially (Joachims et al., 2009a).

One practical approach to train both the "n-slack" and "1-slack" formulations of the SSVMs is a cutting-plane method (Joachims et al., 2009a,b), which is a general optimisation technique for constrained optimisation problems with convex objective and constraints (Boyd and Vandenberghe, 2008). Intuitively, it starts with finding an optimal solution by optimising the objective without any constraints, then generating cutting planes (i.e., linear constraints) by querying a cutting-plane oracle (Wulff and Ong, 2013). This procedure is repeated until convergence. In the next section, we review methods to train the SSVMs, in particular, how one can train the "n-slack" and "1-slack" SSVMs efficiently using cutting-plane methods. More details of cutting-plane methods can be found in Appendix A.

2.3.2 Methods to train the SSVMs

Suppose the affinity function $f(\mathbf{x}, \mathbf{y})$ takes the form of a linear function, i.e., $f(\mathbf{x}, \mathbf{y}) = \mathbf{w}^\top \Psi(\mathbf{x}, \mathbf{y})$ where $\mathbf{w}$ is the parameter vector and $\Psi(\mathbf{x}, \mathbf{y})$ is known as the *joint feature map* of example $\mathbf{x}$ and structured label $\mathbf{y}$.

Training the n-slack SSVMs To train the "n-slack" SSVMs using cutting-plane methods, we make use of a standard QP solver by repeatedly solving a QP with respect to an increasingly larger set of constraints. In each iteration, a new constraint (or cut) is generated to reduce the feasible region of the problem, until the optimal solution achieves a specified precision ε (Joachims et al., 2009b).

Algorithm 1 describes the pseudo-code of a cutting-plane algorithm to train the "n-slack" SSVMs. In each iteration, and for each example, we solve a QP with the current set of constraints $\mathcal{W}$, the solution is our query point $q^{(k)}$ (Line 5). To check if the current query point is ε-feasible, we first compute the most violated label by doing the loss augmented inference (Line 7), then check if $q^{(k)}$ is ε-feasible for this example (Line 8), if not, we generate a feasibility cut (A.3) and add it to the working set $\mathcal{W}$ (Line 9). This procedure is repeated until a ε-feasible query point is found.

As a remark, in Algorithm 1 we can compute the structured hinge loss on the fly for each example (Tsochantaridis et al., 2004), i.e.,

$$\xi_i^{(k)} = \max\left(0, \max_{\bar{\mathbf{y}} \in \mathcal{S}_i} \left\{\Delta(\mathbf{y}^{(i)}, \bar{\mathbf{y}}) + \langle \mathbf{w}^{(k)}, \Psi(\mathbf{x}^{(i)}, \bar{\mathbf{y}})\rangle\right\} - \langle \mathbf{w}^{(k)}, \Psi(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\rangle\right),$$

in this case, the query point $q^{(k)} = \mathbf{w}^{(k)}$.

Training the 1-slack SSVMs The "1-slack" SSVMs can be trained similarly as the "n-slack" SSVMs using cutting-plane methods. Here we replace the N cutting-plane

Algorithm 1 A cutting-plane algorithm to train the n-slack SSVMs

Input: $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{N}$, C, ε
$\mathcal{W} = \emptyset$, $\mathcal{S}_i = \emptyset$, $i \in \{1, \ldots, N\}$, $k = 1$
repeat
  for $i = 1, \ldots, N$ **do**
    Generate query point $q^{(k)} = (\mathbf{w}^{(k)}, \boldsymbol{\xi}^{(k)})$ by solving QP (2.8) w.r.t. all constraints in $\mathcal{W}$
    ▷ Query the oracle at point $q^{(k)}$ as follows
    Do loss-augmented inference: $\hat{\mathbf{y}}^{(k)} = \operatorname{argmax}_{\bar{\mathbf{y}} \in \mathcal{Y}} \{\Delta(\mathbf{y}^{(i)}, \bar{\mathbf{y}}) + \langle \mathbf{w}^{(k)}, \Psi(\mathbf{x}^{(i)}, \bar{\mathbf{y}}) \rangle\}$
    if $q^{(k)}$ is not ε-feasible: $\langle \mathbf{w}^{(k)}, \Psi(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) - \Psi(\mathbf{x}^{(i)}, \hat{\mathbf{y}}^{(k)}) \rangle + \varepsilon < \Delta(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(k)}) - \xi_i^{(k)}$ **then**
      Form a *feasibility cut* and update constraints:
      $\mathcal{W} = \mathcal{W} \cup \left\{ \langle \mathbf{w}, \Psi(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) - \Psi(\mathbf{x}^{(i)}, \hat{\mathbf{y}}^{(k)}) \rangle \geq \Delta(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(k)}) - \xi_i \right\}$
      $\mathcal{S}_i = \mathcal{S}_i \cup \{\hat{\mathbf{y}}^{(k)}\}$, $k = k + 1$
    end if
  end for
until $q^{(k)}$ is ε-feasible for all training examples
return $q^{(k)}$

Algorithm 2 A cutting-plane algorithm to train the 1-slack SSVMs

Input: $S = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$, C, ε
$\mathcal{W} = \emptyset$
for $k = 1, \ldots, +\infty$ **do**
  Generate query point $q^{(k)} = (\mathbf{w}^{(k)}, \xi^{(k)})$ by solving QP (2.11) w.r.t. all constraints in $\mathcal{W}$
  ▷ Query the oracle at point $q^{(k)}$ as follows
  Do loss-augmented inference: $\hat{\mathbf{y}}_i^{(k)} = \operatorname{argmax}_{\bar{\mathbf{y}} \in \mathcal{Y}} \left\{ \Delta(\mathbf{y}^{(i)}, \bar{\mathbf{y}}) + \langle \mathbf{w}^{(k)}, \Psi(\mathbf{x}^{(i)}, \bar{\mathbf{y}}) \rangle \right\}, \forall i$
  if $q^{(k)}$ is ε-feasible, i.e., $\frac{1}{N} \sum_{i=1}^{N} \langle \mathbf{w}^{(k)}, \Psi(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) - \Psi(\mathbf{x}^{(i)}, \hat{\mathbf{y}}_i^{(k)}) \rangle + \varepsilon \geq \frac{1}{N} \sum_{i=1}^{N} \Delta(\mathbf{y}^{(i)}, \hat{\mathbf{y}}_i^{(k)}) - \xi^{(k)}$ **then**
    return $q^{(k)}$
  else
    Form a *feasibility cut* and update constraints:
    $\mathcal{W} = \mathcal{W} \cup \left\{ \frac{1}{N} \sum_{i=1}^{N} \langle \mathbf{w}, \Psi(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) - \Psi(\mathbf{x}^{(i)}, \hat{\mathbf{y}}_i^{(k)}) \rangle \geq \frac{1}{N} \sum_{i=1}^{N} \Delta(\mathbf{y}^{(i)}, \hat{\mathbf{y}}_i^{(k)}) - \xi \right\}$
  end if
end for

models (i.e., one cutting plane for each training example) with a single cutting-plane for the training set in each iteration.

Algorithm 2 describes the pseudo-code of a cutting-plane algorithm to train the

"1-slack" SSVMs. Similar to Algorithm 1, it generates a query point by solving a QP with respect to the working set $\mathcal{W}$ (Line 4). However, to check whether the query point $q^{(k)}$ is ε-feasible, it does the loss-augmented inference to find the most violated label for every example in the training set (Line 6), and then check feasibility (Line 7). If $q^{(k)}$ is ε-feasible, we are done and return $q^{(k)}$ (Line 8), otherwise, we form a feasibility cut (A.3) and add it to the working set $\mathcal{W}$ (Line 10). This procedure is repeated until a ε-feasible query point is found.

As a remark, it generally becomes more difficult to solve a QP when the set of constraints becomes larger, Müller (2014) found that removing the least recently active constraints (i.e., cuts) from the working set $\mathcal{W}$ can speed up the training of SSVMs.

Efficient training via dualisation Recall that in Algorithm 1 and 2, the separation oracle solves a loss-augmented inference problem for each query point and each example, which significantly reduces the scalability of the training algorithm. Techniques have been developed to overcome this repeated inference, typically by exploring the dual problem of either the loss-augmented inference or the structured hinge loss (Taskar, 2004; Taskar et al., 2005; Meshi et al., 2010; Bach et al., 2015).

In particular, if we can find a *concise* formulation of the loss-augmented inference (i.e., the number of variables and constraints in this formulation is *polynomial* in L_i, the number of variables in $\mathbf{y}^{(i)}$), we can write its Lagrangian dual problem as

$$\begin{aligned} \min_{\boldsymbol{\lambda}_i \geq \mathbf{0}} \quad & h_i(\mathbf{w}, \boldsymbol{\lambda}_i) \\ s.t. \quad & g_i(\mathbf{w}, \boldsymbol{\lambda}_i) \leq 0, \end{aligned}$$

where $h_i(\cdot)$ and $g_i(\cdot)$ are convex in both $\mathbf{w}$ and $\boldsymbol{\lambda}_i$. Combining this minimisation over $\boldsymbol{\lambda}_i$ with the minimisation over $\mathbf{w}$ and $\boldsymbol{\xi}$, we have a joint and compact convex minimisation problem (Taskar et al., 2005),

$$\begin{aligned} \min_{\mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\lambda}} \quad & \frac{1}{2}\mathbf{w}^\top \mathbf{w} + \frac{C}{N}\sum_{i=1}^{N} \xi_i \\ s.t. \quad & \mathbf{w}^\top \Psi(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) + \xi_i \geq h_i(\mathbf{w}, \boldsymbol{\lambda}_i), \ \forall i \\ & g_i(\mathbf{w}, \boldsymbol{\lambda}_i) \leq 0, \ \forall i \\ & \boldsymbol{\xi} \geq \mathbf{0}, \ \boldsymbol{\lambda}_i \geq \mathbf{0}, \ \forall i. \end{aligned} \tag{2.13}$$

If $h_i(\cdot)$ and $g_i(\cdot)$ are linear functions of $\mathbf{w}$ and $\boldsymbol{\lambda}_i$, then Problem (2.13) is a quadratic program with polynomial number of variables and constraints, which can be solved using existing off-the-shelf QP solvers.

Alternative training methods Besides the cutting-plane methods, there are a few alternatives that have been developed to train the SSVMs, such as the sub-gradient method (Ratliff et al., 2006) and Frank-Wolfe method (Lacoste-julien et al., 2013).

Recall that the objective of the SSVMs is

$$\frac{1}{2}\mathbf{w}^\top\mathbf{w} + \frac{1}{N}\sum_{i=1}^{N}\max\left(0,\ \max_{\bar{\mathbf{y}}\in\mathcal{Y}}\left\{\Delta(\mathbf{y}^{(i)},\bar{\mathbf{y}}) - \left(f(\mathbf{x}^{(i)},\mathbf{y}^{(i)}) - f(\mathbf{x}^{(i)},\bar{\mathbf{y}})\right)\right\}\right),$$

observing that the structured hinge loss in the objective is not differentiable, however, its sub-gradient with respect to $\mathbf{w}$ still exists, the sub-gradient method optimises this objective by leveraging its sub-gradient to solve an unconstrained optimisation problem. The Frank-Wolfe method optimises the first-order linear approximation of the quadratic objective in (2.8). It transforms the quadratic program into a series of linear programs that can be solved more efficiently. In practice, these methods can sometimes converge faster than cutting-plane methods.

To conclude, we have described a number of methods to efficiently train the SSVMs, in the next section, we review efficient inference techniques for prediction given a trained SSVMs, in particular those that can decode paths in Markov chains.

2.4 Path decoding in Markov chains

In this section, we first review the forward and backward approaches that solve two typical problems in hidden Markov models (HMMs), i.e., computing the probability of an observed sequence, and identifying the most likely sequence of states for a given observed sequence. These methods are better known as the forward-backward algorithm and the Viterbi algorithm (and its backward variant). Interestingly, these algorithms are closely related in the sense that they are different special cases of the generalised distributive law (Aji and McEliece, 2000). Further, we review methods that generalise either the Viterbi algorithm or its backward variant to find K most likely state sequences for an observed sequence, which are known as the list Viterbi algorithm (LVA). We leverage the LVA to solve the path decoding problem in Markov chains, i.e., finding the most likely path (i.e., state sequence without repeated states). Finally, we review another approach for the path decoding problem based on the integer linear programming formulation of the travelling salesman problem (TSP), as well as a few heuristic algorithms. In Chapter 4, we show how these path decoding techniques can be employed to address essential challenges in recommending paths, e.g., in the task of recommending travel trajectories.

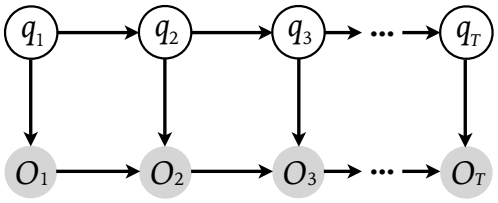


Figure 2.3: Graphical model of a hidden Markov model (HMM). The observed sequence $O_1O_2\cdots O_T$ is generated by a sequence of T hidden states $q_1q_2\cdots q_T$.

2.4.1 The forward and backward approaches in HMM

Given a hidden Markov model (HMM) with N states and parameters $\theta = (A, B, \pi)$, $A = \{a_{ij}\}$, $B = \{b_j(v_k)\}$, $\pi = \{\pi_i\}$, state sequence $\mathbf{q} = q_{1:T} = q_1q_2\cdots q_T$, $q_t \in \{s_i\}_{i=1}^N$, and observation sequence $O_{1:T} = O_1O_2\cdots O_T$, $O_t \in \{v_m\}_{m=1}^M$, as illustrated in Figure 2.3, where

$$\begin{aligned} a_{ij} &\doteq \mathbb{P}(q_{t+1} = s_j \mid q_t = s_i), && i, j \in \{1, \ldots, N\} \\ b_j(v_m) &\doteq \mathbb{P}(v_m \mid s_j), && 1 \le j \le N, 1 \le m \le M \\ \pi_i &\doteq \mathbb{P}(q_1 = s_i), && 1 \le i \le N. \end{aligned} \tag{2.14}$$

The forward-backward algorithms solve the problem of computing the likelihood of an observed sequence $O_{1:T}$, and the Viterbi algorithm and its backward variant deal with the problem of identifying the most likely sequence of states for an observed sequence $O_{1:T}$. Interestingly, both problems can be computed either forwards or backwards along the observation sequence by making use of the conditional independences encoded in an HMM:

$$\begin{aligned} q_t &\perp\!\!\!\perp q_{k\notin\{t-1,t\}} \mid q_{t-1}, && \forall t \in \{2, \ldots, T\}, \\ q_t &\perp\!\!\!\perp O_{1:T} \mid q_{t-1}, && \forall t \in \{2, \ldots, T\}, \\ O_t &\perp\!\!\!\perp q_{k\neq t} \mid q_t, && \forall t \in \{1, \ldots, T\}, \\ O_t &\perp\!\!\!\perp O_{k\neq t} \mid q_t, && \forall t \in \{1, \ldots, T\}. \end{aligned} \tag{2.15}$$

We first describe the forward approach of likelihood computation and sequence identification, followed by the backward approach for these two problems.

The forward algorithm

The forward algorithm is based on the insight that the likelihood of observed sequence $O_{1:T}$ can be computed from the likelihood of the sub-sequence $O_{1:T-1}$, i.e.,

$$\mathbb{P}(O_{1:T}; \theta) = \sum_i \mathbb{P}(O_{1:T}, q_T = s_i; \theta),$$

and observe that

$$\begin{aligned}\mathbb{P}(O_{1:T}, q_T = s_i; \theta) = &\left[\sum_j \mathbb{P}(O_{1:T-1}, q_{T-1} = s_j; \theta) \cdot \mathbb{P}(q_T = s_i \mid q_{T-1} = s_j)\right] \\ &\cdot \mathbb{P}(O_T \mid q_T = s_i),\end{aligned} \tag{2.16}$$

where for simplicity, we define

$$\mathbb{P}(O_{1:t}, q_t = s_i; \theta) \doteq \sum_{q_{1:t-1}} \mathbb{P}(O_{1:t}, q_{1:t-1}, q_t = s_i; \theta).$$

Let
$$\alpha_t(s_i) = \mathbb{P}(O_{1:t}, q_t = s_i; \theta). \tag{2.17}$$

Then by Eq. (2.14) and (2.16) the likelihood of $O_{1:T}$ is

$$\begin{aligned}\mathbb{P}(O_{1:T}; \theta) &= \sum_i \alpha_T(s_i) \\ &= \sum_i \left[\sum_j \alpha_{T-1}(s_j) \cdot a_{ji}\right] b_i(O_T).\end{aligned} \tag{2.18}$$

We can repeat the procedure in (2.16) to decompose $\alpha_{T-1}(s_j)$, in general, we have

$$\alpha_t(s_i) = \begin{cases} \pi_i \cdot b_i(O_t), & t = 1, \\ \left[\sum_j \alpha_{t-1}(s_j) \cdot a_{ji}\right] b_i(O_t), & t = 2, \ldots, T. \end{cases} \tag{2.19}$$

Computing the $\alpha_t(s_i)$ values by Eq. (2.19) is known as the *forward algorithm*, and the likelihood of the observated sequence $O_{1:T}$ is $\sum_i \alpha_T(s_i)$.

The Viterbi algorithm

To identify the most likely sequence of states for an observed sequence $O_{1:T}$, note that

$$q^*_{1:T} = \operatorname*{argmax}_{q_{1:T}} \mathbb{P}(O_{1:T}, q_{1:T}; \theta).$$

Further, similar to (2.16), we observe that the computation can be decomposed into smaller sub-problems

$$\begin{aligned}\max_{q_{1:T}} \mathbb{P}(O_{1:T}, q_{1:T}; \theta) &= \max_i \left\{\max_{q_{1:T-1}} \mathbb{P}(O_{1:T}, q_{1:T-1}, q_T = s_i; \theta)\right\}, \\ \max_{q_{1:T-1}} \mathbb{P}(O_{1:T}, q_{1:T-1}, q_T = s_i; \theta) &= \max_j \left\{\max_{q_{1:T-2}} \mathbb{P}(O_{1:T-1}, q_{1:T-2}, q_{T-1} = s_j; \theta)\right. \\ &\qquad \left. \cdot \mathbb{P}(q_T = s_i \mid q_{T-1} = s_j)\right\} \cdot \mathbb{P}(O_T \mid q_T = s_i).\end{aligned} \tag{2.20}$$

Let $$\widetilde{\alpha}_t(s_i) = \max_{q_{1:t-1}} \mathbb{P}(O_{1:t}, q_{1:t-1}, q_t = s_i; \theta), \tag{2.21}$$

by Eq. (2.14) and (2.20) we have

$$\max_{q_{1:T}} \mathbb{P}(O_{1:T}, q_{1:T}; \theta) = \max_i \widetilde{\alpha}_T(s_i) = \max_i \left\{ \max_j \left\{ \widetilde{\alpha}_{T-1}(s_j) \cdot a_{ji} \right\} b_i(O_T) \right\}. \tag{2.22}$$

We can repeat the procedure in (2.20) to decompose $\widetilde{\alpha}_{T-1}(s_j)$, in general, we have

$$\widetilde{\alpha}_t(s_i) = \begin{cases} \pi_i \cdot b_i(O_t), & t = 1, \\ \left[\max_j \widetilde{\alpha}_{t-1}(s_j) \cdot a_{ji}\right] b_i(O_t), & t = 2, \ldots, T. \end{cases} \tag{2.23}$$

Computing the $\widetilde{\alpha}_t(s_i)$ values by Eq. (2.23) is known as the *Viterbi algorithm*, and a sequence $\mathbf{q}^* = q^*_{1:T}$ with highest probability can be identified via back-tracking.

As a remark, the only difference between the forward algorithm (Eq. 2.19) and the Viterbi algorithm (Eq. 2.23) is that the former uses *summation* and the latter uses *maximisation* (Aji and McEliece, 2000). Computations in Eq. (2.18) and (2.22) can be simplified by adding dummy (*deterministic*) state $q_{T+1} = s^*_{T+1}$ and observation O_{T+1},

$$\mathbb{P}(O_{1:T}; \theta) = \sum_{q_{1:T}} \mathbb{P}(O_{1:T+1}, q_{1:T}, q_{T+1} = s^*_{T+1}; \theta) = \alpha_{T+1}(s^*_{T+1})$$
$$\max_{q_{1:T}} \mathbb{P}(O_{1:T}, q_{1:T}; \theta) = \max_{q_{1:T}} \mathbb{P}(O_{1:T+1}, q_{1:T}, q_{T+1} = s^*_{T+1}; \theta) = \widetilde{\alpha}_{T+1}(s^*_{T+1}).$$

The backward algorithm

Alternatively, the likelihood of an observed sequence $O_{1:T}$ can be computed backwards, note that $\mathbb{P}(O_{1:T}; \theta) = \sum_{q_{1:T}} \mathbb{P}(O_{1:T}, q_{1:T}; \theta)$, and by Eq. (2.15),

$$\begin{aligned} \sum_{q_{1:T}} \mathbb{P}(O_{1:T}, q_{1:T}; \theta) &= \sum_i \sum_{q_{2:T}} \mathbb{P}(q_1 = s_i, q_{2:T}, O_{1:T}; \theta) \\ &= \sum_i \left[\sum_{q_{2:T}} \mathbb{P}(q_{2:T}, O_{2:T} \mid q_1 = s_i; \theta) \right] \mathbb{P}(q_1 = s_i) \cdot \mathbb{P}(O_1 \mid q_1 = s_i) \\ \sum_{q_{2:T}} \mathbb{P}(q_{2:T}, O_{2:T} \mid q_1 = s_i; \theta) &= \sum_j \left[\sum_{q_{3:T}} \mathbb{P}(q_{3:T}, O_{3:T} \mid q_2 = s_j; \theta) \right] \mathbb{P}(q_2 = s_j \mid q_1 = s_i) \cdot \mathbb{P}(O_2 \mid q_2 = s_j) \end{aligned} \tag{2.24}$$

Let $$\beta_t(s_i) = \sum_{q_{t+1:T}} \mathbb{P}(q_{t+1:T}, O_{t+1:T} \mid q_t = s_i; \theta), \tag{2.25}$$

and by Eq. (2.14) and (2.24) we have

$$\mathbb{P}(O_{1:T}; \theta) = \sum_i \beta_1(s_i) \cdot \pi_i \cdot b_i(O_1) = \sum_i \left[\sum_j \beta_2(s_j) \cdot a_{ij} \cdot b_j(O_2) \right] \pi_i \cdot b_i(O_1). \tag{2.26}$$

We can repeat the procedure in (2.24) to decompose $\beta_2(s_j)$, in general, we have

$$\beta_t(s_i) = \begin{cases} 1, & t = T \\ \sum_j \beta_{t+1}(s_j) \cdot a_{ij} \cdot b_j(O_{t+1}), & t = T-1 \ \textbf{downto} \ 1. \end{cases} \tag{2.27}$$

Computing the $\beta_t(s_i)$ values by Eq. (2.27) is known as the *backward algorithm*, and the likelihood of observation sequence $O_{1:T}$ is $\sum_i \beta_1(s_i) \cdot \pi_i \cdot b_i(O_1)$.

The "backward Viterbi" algorithm

We can also identify the most likely sequence of states for observations $O_{1:T}$ by computing backwards similar to (2.27), note that

$$\begin{aligned} \max_{q_{1:T}} \mathbb{P}(O_{1:T}, q_{1:T}; \theta) &= \max_i \max_{q_{2:T}} \mathbb{P}(q_1 = s_i, q_{2:T}, O_{1:T}; \theta) \\ &= \max_i \left\{ \max_{q_{2:T}} \mathbb{P}(q_{2:T}, O_{2:T} \mid q_1 = s_i; \theta) \right\} \mathbb{P}(q_1 = s_i) \cdot \mathbb{P}(O_1 \mid q_1 = s_i), \end{aligned} \tag{2.28}$$

and by the conditional independences (2.15), we have

$$\begin{aligned} \max_{q_{2:T}} \mathbb{P}(q_{2:T}, O_{2:T} \mid q_1 = s_i; \theta) &= \max_j \max_{q_{3:T}} \mathbb{P}(q_2 = s_j, q_{3:T}, O_{2:T} \mid q_1 = s_i; \theta) \\ &= \max_j \left\{ \max_{q_{3:T}} \mathbb{P}(q_{3:T}, O_{3:T} \mid q_2 = s_j; \theta) \right\} \\ &\quad \cdot \mathbb{P}(q_2 = s_j \mid q_1 = s_i) \cdot \mathbb{P}(O_2 \mid q_2 = s_j). \end{aligned} \tag{2.29}$$

Let

$$\widetilde{\beta}_t(s_i) = \max_{q_{t+1:T}} \mathbb{P}(q_{t+1:T}, O_{t+1:T} \mid q_t = s_i; \theta), \tag{2.30}$$

by Eq. (2.14), (2.28) and (2.29) we have

$$\begin{aligned} \max_{q_{1:T}} \mathbb{P}(O_{1:T}, q_{1:T}; \theta) &= \max_i \widetilde{\beta}_1(s_i) \cdot \pi_i \cdot b_i(O_1) \\ &= \max_i \left\{ \max_j \widetilde{\beta}_2(s_j) \cdot a_{ij} \cdot b_j(O_2) \right\} \pi_i \cdot b_i(O_1). \end{aligned} \tag{2.31}$$

We can repeat the procedure in (2.29) to decompose $\widetilde{\beta}_2(s_j)$, in general, we have

$$\widetilde{\beta}_t(s_i) = \begin{cases} 1, & t = T \\ \max_j \widetilde{\beta}_{t+1}(s_j) \cdot a_{ij} \cdot b_j(O_{t+1}), & t = T-1 \ \textbf{downto} \ 1. \end{cases} \tag{2.32}$$

Interestingly, if we compare the Viterbi algorithm (Eq. 2.23) with Eq. (2.32), we note that the latter can be treated as the Viterbi algorithm works backwards, and a sequence

$q^*_{1:T}$ with highest probability can be identified via *forward*-tracking.

As a remark, the only difference between Eq. (2.27) and (2.32) is that the former uses *summation* and the latter uses *maximisation* (Aji and McEliece, 2000). Computations in Eq. (2.26) and (2.31) can also be simplified by adding dummy (*deterministic*) state $q_0 = s^*_0$ and observation O_0,

$$\mathbb{P}(O_{1:T}; \theta) = \sum_{q_{1:T}} \mathbb{P}(O_{0:T}, q_0 = s^*_0, q_{1:T}; \theta) = \sum_{q_{1:T}} \mathbb{P}(O_{1:T}, q_{1:T} \mid q_0 = s^*_0; \theta) = \beta_0(s^*_0)$$

$$\max_{q_{1:T}} \mathbb{P}(O_{1:T}, q_{1:T}; \theta) = \max_{q_{1:T}} \mathbb{P}(O_{0:T}, q_0 = s^*_0, q_{1:T}; \theta) = \max_{q_{1:T}} \mathbb{P}(O_{1:T}, q_{1:T} \mid q_0 = s^*_0; \theta) = \widetilde{\beta}_0(s^*_0).$$

2.4.2 The list Viterbi algorithm

For tasks such as decoding digital signals corrupted by noise (e.g., speech recognition), it has been shown that considerable improvement can be obtained when a list of top $K > 1$ best sequences through the HMM trellis graph can be utilised (Ostendorf et al., 1991; Seshadri and Sundberg, 1994; Nill and Sundberg, 1995; Stolcke et al., 1997). In this section, we describe algorithms that generalise the Viterbi algorithm to find the top K best sequences from a HMM, which are known as the *list Viterbi algorithm* (Nill and Sundberg, 1995; Nilsson and Goldberger, 2001).

The list Viterbi algorithm (LVA) can be employed to decode paths in a HMM by sequentially searching the next best sequence through the trellis graph until the specified number of paths are found. In Chapter 4, we make use of the LVA to deal with two challenges for path recommendations: incorporating multiple ground truths in training and eliminating loops in prediction.

Let $\gamma_t(s_i, s_j)$ denote the likelihood of an observed sequence $O_{1:T}$ such that the states at time t and $t+1$ are s_i and s_j respectively. By Eq. (2.17) and (2.25) we have

$$\begin{aligned}
\gamma_t(s_i, s_j) &= \sum_{q_{1:t-1, t+2:T}} \mathbb{P}(O_{1:T}, q_{1:t-1}, q_t = s_i, q_{t+1} = s_j, q_{t+2:T}; \theta) \\
&= \left[\sum_{q_{1:t-1}} \mathbb{P}(O_{1:t}, q_{1:t-1}, q_t = s_i; \theta) \right] \mathbb{P}(q_{t+1} = s_j \mid q_t = s_i) \cdot \mathbb{P}(O_{t+1} \mid q_{t+1} = s_j) \\
&\quad \cdot \left[\sum_{q_{t+2:T}} \mathbb{P}(O_{t+2:T}, q_{t+2:T} \mid q_{t+1} = s_j; \theta) \right] \\
&= \alpha_t(s_i) \cdot a_{ij} \cdot b_j(O_{t+1}) \cdot \beta_{t+1}(s_j).
\end{aligned} \tag{2.33}$$

We note that $\gamma_t(s_i, s_j)$ is the *marginal* joint probability of all possible pairs of observations-states $(O_{1:T},\ q_{1:t-1}\, s_i\, s_j\, q_{t+2:T})$ given an HMM. Similar to Section 2.4.1, we can compute the *maximum* joint probability over all possible pairs of observations-states

$(O_{1:T},\ q_{1:t-1}\, s_i\, s_j\, q_{t+2:T})$, by Eq. (2.15), (2.21) and (2.30) we have

$$\begin{aligned}\widetilde{\gamma}_t(s_i, s_j) &= \max_{q_{1:t-1,t+2:T}} \mathbb{P}(O_{1:T}, q_{1:t-1}, q_t = s_i, q_{t+1} = s_j, q_{t+2:T}; \theta) \\ &= \left\{ \max_{q_{1:t-1}} \mathbb{P}(O_{1:t}, q_{1:t-1}, q_t = s_i; \theta) \right\} \mathbb{P}(q_{t+1} = s_j \mid q_t = s_i) \cdot \mathbb{P}(O_{t+1} \mid q_{t+1} = s_j) \\ &\quad \cdot \left\{ \max_{q_{t+2:T}} \mathbb{P}(O_{t+2:T}, q_{t+2:T} \mid q_{t+1} = s_j; \theta) \right\} \\ &= \widetilde{\alpha}_t(s_i) \cdot a_{ij} \cdot b_j(O_{t+1}) \cdot \widetilde{\beta}_{t+1}(s_j).\end{aligned} \tag{2.34}$$

Nilsson and Goldberger (2001) proposed an algorithm to find $K(K > 1)$ most likely state sequences for a given observed sequence $O_{1:T}$ using information encoded in $\widetilde{\gamma}_t$ (Eq. 2.34). It cleverly partitions the space of possible state sequences into subsets and identifies the most likely state sequence in each subset, this procedure is repeated until all K most likely state sequences have been found.

Algorithm 3 shows the pseudocode of the method of Nilsson and Goldberger (2001), it uses a max-heap to keep track of the sequence with the highest probability in each subset (Line 5, 7, and 14). For a sequence with the highest probability (in a subset), the algorithm maintains a partition index (i.e., the time instant at which the sequence diverges from the previous best sequence, Line 9) and an excluding set (i.e., a set of states that should be excluded at the time instant indicated by the partition index when computing the most likely state sequence in the partitioned subset, Line 11). A property of $\widetilde{\gamma}$ (Nilsson and Goldberger, 2001, Theorem 1) has been employed to efficiently compute the probability of the most likely state sequence in a subset as well as to identify the state sequence itself (Line 12 to Line 13).

Comparison of LVA variants

We compare three typical variants of the LVA (i.e., parallel LVA, serial LVA and Algorithm 3) in terms of their time and space complexity.

The *parallel LVA* (Seshadri and Sundberg, 1994) finds the K best state sequences simultaneously by computing the K best state sequences in each state at each time instant. The idea is caching all the K best scores (i.e., probabilities) at each time instant instead of the best score as that in the Viterbi algorithm. This needs to find the K best scores among the NK accumulated scores at each state and at each time instant (Seshadri and Sundberg, 1994, Eq. 6), which can be achieved by sorting the NK accumulated scores in time $O(NK \log(NK))$, or using an order statistics selection algorithm (Cormen et al., 2009, Chapter 9) with time complexity $O(K \cdot NK)$. Since the above step is carried out NT times, the time complexity of the parallel LVA is

Algorithm 3 The Nilsson and Goldberger (2001) list Viterbi algorithm

Input: HMM parameters θ, observations $O_{1:T}$, sequence length T, the number of most likely sequences K.
Initialise max-heap H, result set $R = \emptyset$, $k = 1$.
Compute $\widetilde{\alpha}_t(s_i)$ (Eq. 2.21), $\widetilde{\beta}_t(s_i)$ (Eq. 2.30) and $\widetilde{\gamma}_t(s_i, s_j)$ (Eq. 2.34).
Identify a sequence with the highest probability (Eq. 2.23) via back-tracking: let $\mathbf{q}^{[1]} = (q_1^{[1]}, \ldots, q_T^{[1]})$ be the sequence and $r^{[1]}$ be the corresponding probability.
$H.push(r^{[1]},\ (\mathbf{q}^{[1]}, \textsc{nil}, \emptyset))$
while $H \neq \emptyset$ **and** $k \leq K$ **do**
  $r^{[k]},\ (\mathbf{q}^{[k]}, I, S) = H.pop()$
  Extract a state sequence with the k-th highest probability from the max-heap H: $r^{[k]}$ is the probability of $\mathbf{q}^{[k]} = (q_1^{[k]}, \ldots, q_T^{[k]})$, I is the partition index, and S is the excluding set.
  Add $\mathbf{q}^{[k]}$ to the result set R.
  $$I' = \begin{cases} 1, & I = \textsc{nil} \\ I, & \text{otherwise} \end{cases}$$
  for $t = I'$ **to** T **do**
    $$S' = \begin{cases} S \cup \{q_t^{[k]}\}, & t = I' \\ \{q_t^{[k]}\}, & \text{otherwise} \end{cases}$$
    $$q_{t'} = \begin{cases} q_{t'}^{[k]}, & t' = 1 \textbf{ to } t-1 \\ \underset{s \notin S'}{\operatorname{argmax}} \left\{ \widetilde{\gamma}_{t-1,t}(q_{t-1}^{[k]}, s) \right\}, & t' = t \\ \underset{s}{\operatorname{argmax}} \left\{ \widetilde{\gamma}_{t'-1,t'}(q_{t'-1}, s) \right\}, & t' = t+1 \textbf{ to } T \end{cases}$$
    $$r = r^{[k]} \cdot \frac{\widetilde{\gamma}_{t-1,t}(q_{t-1}^{[k]}, q_t)}{\widetilde{\gamma}_{t-1,t}(q_{t-1}^{[k]}, q_t^{[k]})}$$ (Nilsson and Goldberger, 2001, Theorem 1)
    $H.push(r, (\mathbf{q}, t, S'))$
  end for
  $k = k + 1$
end while
return result set R

$O(N^2KT\log(NK))$ or $O(N^2K^2T)$. To analyse the space complexity, note that we need $O(NKT)$ space to store all sequence states, and $O(NK)$ space for the NK accumulated scores at a time instant. Therefore, the space complexity of parallel LVA is $O(NKT)$.

The *serial LVA* (Seshadri and Sundberg, 1994) finds the K best state sequences one at a time. It leverage the insight that the globally second best state sequence diverges from the globally first best sequence at some time instant, and it merges with the

globally first best sequence at a later time instant and never diverges again, since any subsequent divergence will result in a higher cost (lower score) sequence. Similarly, the third globally best sequence is either the locally third best sequence with respect to the globally first best sequence (i.e., excluding the globally second best sequence) or the locally second best sequences with respect to the globally second best sequence. This reasoning can be generalised to find the K-th best state sequence given the first, second, ..., $(K-1)$-th best state sequences.

To analyse the time complexity of the serial LVA, note that

- computing the globally best sequence using the Viterbi algorithm in time (N^2T), i.e., filling in a table with NT cells where each can be computed in $O(N)$ time.
- computing the locally second best sequences after identifying the globally k-th $(k = 1, \ldots, K-1)$ best sequences. This requires N additions and comparisons in (Seshadri and Sundberg, 1994, Eq. 9) for each time instant $t \in \{1, \ldots, T\}$, which can be upper bounded by $O(NT)$. Thus, for all K best sequences, the time complexity is $O(NKT)$.
- Suppose we use a priority queue to keep track of the locally best sequences, the additional cost related to *insertion* and *extracting best* operations can be upper bounded by $O(KT \log(KT))$. In other words, the total number of sequences in the priority queue is KT at most, and the time for each insertion or extraction can be upper bounded by $\log(KT)$.[4]

Adding up these terms, the time complexity of the serial LVA is $O(N^2T + NKT + KT\log(KT))$, and the space complexity is $\Omega(NT + KT + NTK)$ i.e., the path matrix and the "merge" count array (Seshadri and Sundberg, 1994, Eq. 9).

The analysis of the time complexity of Algorithm 3 is similar to that of the serial LVA. First, the cost of the Viterbi and "backward Viterbi" algorithms is $O(N^2T)$, then the partitioning operations and the computation of the best sequence in each partition cost $O(K \cdot NT)$. Lastly, the insertion and extraction of sequences cost $O(KT\log(KT))$ if we use a priority queue. This results in $O(N^2T + NKT + KT\log(KT))$. The space complexity of Algorithm 3 is

$$O(NT + KT \cdot (1 + T + 1 + N)) = O(KT^2 + NKT).$$

As a remark, the parallel LVA needs to specify the value of K as an input of the algorithm, however, in practice, we usually do not know the value of K beforehand.

[4] A tighter bound is $O\left(\sum_{k=1}^{K} \log(kT)\right) = O(K\log(T) + \log(K!))$, which can be further simplified using the Stirling's formula (Dutka, 1991).

Table 2.2: Time and space complexities of three typical list Viterbi algorithms.

LVA variants	Time complexity	Space complexity
Parallel LVA	$O\left(N^2KT\log(NK)\right)$ or $O\left(N^2K^2T\right)$	$O(NKT)$
Serial LVA	$O\left(N^2T + NKT + KT\log(KT)\right)$	$\Omega(NT + KT + NKT)$
Nilsson and Goldberger (2001)	$O\left(N^2T + NKT + KT\log(KT)\right)$	$O(KT^2 + NKT)$

In this case, the serial LVA and Algorithm 3 (Nilsson and Goldberger, 2001) are applicable as both algorithms sequentially find the next-best state sequence.

Table 2.2 summaries the time and space complexity of the three LVA variants.

Although the serial LVA and Algorithm 3 are ostensibly different, it turns out that these two algorithms are in fact computing the same fundamental quantities (Menon, 2017; Menon et al., 2017). In particular, Menon (2017) showed that, similar to the forward and backward approaches described in Section 2.4.1, the only difference between the serial LVA and Algorithm 3 is simply that the former works forwards and the latter works backwards.

2.4.3 Sub-tour elimination in s-t path TSP

Another technique that can decode paths in Markov chains was developed for the travelling salesman problem (TSP). Given a complete graph with N nodes, the travelling salesman problem is to find a shortest route that visits each node exactly once and finishes at the starting node. The s-t path TSP is a variant of TSP that finding a shortest path with fixed endpoints s and t (Hoogeveen, 1991; An et al., 2015).

Let d_{ij} be the distance between node i and node j, the s-t path TSP can be formulated as an integer linear program (ILP):

$$\max_{\mathbf{x},\mathbf{u}} \sum_{i,j} c_{ij} z_{ij}$$

$$s.t.\ \ z_{ij} \in \{0,1\},\ z_{ii} = 0,\ u_i \in \mathbb{Z}, \qquad \forall i,j \in \{1,\dots,N\}$$

$$\sum_{i=1}^{N} z_{i1} = 0,\ \sum_{j=1}^{N} z_{1j} = 1,\ \sum_{i=1}^{N} z_{iN} = 1,\ \sum_{j=1}^{N} z_{Nj} = 0, \tag{2.35}$$

$$\sum_{j=1}^{N} z_{ij} = \sum_{j=1}^{N} z_{ji} \le 1, \qquad \forall i \in \{2,\dots,N-1\} \tag{2.36}$$

$$\sum_{i=1}^{N-1} \sum_{j=2}^{N} z_{ij} = N - 1, \tag{2.37}$$

$$u_i - u_j + 1 \le (N-1)(1 - z_{ij}). \qquad \forall i,j \in \{1,\dots,N\} \tag{2.38}$$

For brevity, here we index the nodes such that s is node 1 and t is node N. The binary variables z_{ij} are true *iff* node i is visited immediately after visiting node j;

integer variable u_i tracks the rank of node i in the shortest path.[5] Constraint (2.35) ensures the path starts at node s and ends at t. Further, no self-loops are allowed (i.e., $z_{ii} = 0$), and each node should be visited exactly once (Constraints 2.36 and 2.37). In particular, Constraint (2.38) ensures we do not have (disjoint) cycles (i.e., *sub-tours*) in the resulting path, as per Miller et al. (1960).

We can adapt the above ILP formulation to compute the most likely path of states for a hidden Markov model given the length of path. In particular, we maximise both the transition likelihood and emission probabilities. However, in order to incorporate observations $O_{1:T}$ in the objective of the ILP formulation, we need $N^2(T-1)$ binary variables (instead of N^2 variables in the ILP formulation for the s-t path TSP) where decision variable $z_{ijt} = 1$ represents that $q_t = s_i$ and $q_{t+1} = s_j$, and constraints (2.35-2.38) should be adapted accordingly. Similar to the LVA, we can find the K most likely paths instead of just one most likely path in a sequential manner, i.e., by encoding the first, the second, ..., the $K-1$-th most likely paths in new constraints when computing the K-th most likely path.

2.4.4 Heuristic algorithms

Besides the list Viterbi algorithm and the approach based on the ILP formulation of the s-t path TSP, there are many heuristic algorithms that solve the problem of decoding path in Markov chains, which do not necessarily return the optimal path but can work efficiently. A straightforward approach is to post-process the sequence decoded by the Viterbi algorithm, e.g., simply skipping a state when it appears for the second time; however, this approach will result in a shorter sequence if the one decoded by the Viterbi algorithm contains duplicate states. To mitigate this problem, Menon et al. (2017) proposed to decoding a longer sequence than required using the Viterbi algorithm before the post-processing.

Another heuristic approach is to decode the sequence using a greedy method. At each decoding step, it can either choose the best state or caching K best states (i.e., the beam search method) to avoid a state being included more than once in the resulting path. Alternatively, if the path decoding problem can be formulated as an instance of the TSP, for example, by fixing a subset of states, then heuristic algorithms of the TSP can also be applied, e.g., the Christofides algorithm[6] which can find a path that has at most 1.5 times the cost of the optimal path in polynomial time (Christofides, 1976).

[5] This is easy to observe if we assume the shortest path is ordered from node π_1 to node π_N, i.e., $z_{\pi_t,\pi_{t+1}} = 1$, $t \in \{1,\dots,N-1\}$, by Constraint (2.38) we have $u_{\pi_{t+1}} \geq u_{\pi_t} + 1$, which means, in the shortest path, the $t+1$-th node is ranked higher (with a margin of at least 1) than the t-th node.

[6] Note that the Christofides algorithm approximately solves the metric-TSP that requires the edge weights in the complete graph to satisfy the triangle inequality.

2.5 Binary classification and bipartite ranking

We have reviewed techniques that form the foundation of efficient path recommendation. In this section, we describe the problems and related techniques that can help us achieve efficient set recommendation. In particular, we review a few closely related loss functions for the problem of binary classification and bipartite ranking, which will be employed in Chapter 5 to enable efficient recommendation of sets in the context of music playlists.

Given a sample of instances (or examples) with binary labels, *binary classification* is the problem of learning a binary-valued function that gives $+1$ labels for positive instances and -1 labels for negative instances.[7] A related problem is *bipartite ranking*, which learns to rank positive instances above negative instances.

Let $\mathcal{X}$ be the instance space, and $\mathcal{D} = \mathcal{S}_+ \cup \mathcal{S}_-$ denote the binary dataset, where $\mathcal{S}_+ = \{(\mathbf{x}_+, +1)\}$ is a set of positive examples, and $\mathcal{S}_- = \{(\mathbf{x}_-, -1)\}$ is a set of negative examples. Examples $\mathbf{x}_+, \mathbf{x}_- \in \mathcal{X}$, e.g., D-dimensional feature vectors. Further, we use M_+ and M_- to denote the number of positive and negative examples, respectively.

2.5.1 Loss functions for binary classification

The binary-valued prediction function for a binary classifier is generally not directly learned, rather, we first learn a real-valued scoring function $f : \mathcal{X} \to \mathbb{R}$, then compare the score of a particular input with a threshold τ to get a binary prediction. The misclassification loss of f on binary dataset $\mathcal{D}$ is the number of incorrectly classified examples. Taking care of ties, we have

$$R^{\text{BC}}_{0/1} = \sum_{\mathbf{x}_+ \in \mathcal{S}_+} \left([\![f(\mathbf{x}_+) < \tau]\!] + \frac{1}{2} [\![f(\mathbf{x}_+) = \tau]\!] \right) + \sum_{\mathbf{x}_- \in \mathcal{S}_-} \left([\![f(\mathbf{x}_-) > \tau]\!] + \frac{1}{2} [\![f(\mathbf{x}_-) = \tau]\!] \right), \tag{2.39}$$

where $[\![\cdot]\!]$ is the indicator function that represents the $0/1$ loss.

To practically optimise the misclassification loss $R^{\text{BC}}_{0/1}$ we replace the indicator function in (2.39) with one of the convex surrogate of the $0/1$ loss to upper bound $R^{\text{BC}}_{0/1}$, in other words, we have

$$R^{\text{BC}}_{0/1} \leq \sum_{\mathbf{x}_+ \in \mathcal{S}_+} \ell(f(\mathbf{x}_+) - \tau) + \sum_{\mathbf{x}_- \in \mathcal{S}_-} \ell(\tau - f(\mathbf{x}_-)).$$

For example, let $\tau = 0$ and $\ell(z) = \max(0, 1 - z)$ (i.e., the hinge loss), we get the empirical risk of the support vector machines for binary classification.[8] Alternatively,

[7] The labels for binary classification are also (widely) denoted as $\{0, 1\}$ or $\{\texttt{True}, \texttt{False}\}$.

[8] As a remark, loss functions for binary classification can be independent of the threshold, a

let $\tau = 0$ and $\ell(z) = e^{-z}$ (i.e., the exponential loss) results in the objective of AdaBoost (Freund and Schapire, 1997), and here we review two generalisations of it. The first one is known as the *Cost-Sensitive AdaBoost* (Ertekin and Rudin, 2011)

$$R^{\text{CSA}}(f, \mathcal{D}) = \sum_{\mathbf{x}_+ \in \mathcal{S}_+} \exp(-f(\mathbf{x}_+)) + C \sum_{\mathbf{x}_- \in \mathcal{S}_-} \exp(f(\mathbf{x}_-)), \tag{2.40}$$

where C is a weighting parameter. Another generalisation of the AdaBoost is the *P-Classification* (Ertekin and Rudin, 2011)

$$R^{\text{PC}}(f, \mathcal{D}) = \sum_{\mathbf{x}_+ \in \mathcal{S}_+} \exp(-f(\mathbf{x}_+)) + \frac{1}{p} \sum_{\mathbf{x}_- \in \mathcal{S}_-} \exp(pf(\mathbf{x}_-)), \tag{2.41}$$

where $p \in \mathbb{R}^+$ is a hyper-parameter.

It turns out that both the Cost-Sensitive AdaBoost (Eq. 2.40) and the P-Classification (Eq. 2.41) are closely related to the P-Norm Push (Eq. 2.43) (Ertekin and Rudin, 2011). This inspires a more general relation between bipartite ranking and binary classification, which will be described in the next section.

2.5.2 Loss functions for bipartite ranking

Given an example space $\mathcal{X}$, let $f : \mathcal{X} \to \mathbb{R}$ be a function that can score an example $\mathbf{x} \in \mathcal{X}$. The misranking loss of f on binary dataset $\mathcal{D}$ is the number of positive examples that are ranked below any negative example. Accounting for ties, we have

$$R^{\text{BR}}_{0/1}(f, \mathcal{D}) = \sum_{\mathbf{x}_+ \in \mathcal{S}_+} \sum_{\mathbf{x}_- \in \mathcal{S}_-} \left([\![f(\mathbf{x}_+) < f(\mathbf{x}_-)]\!] + \frac{1}{2} [\![f(\mathbf{x}_+) = f(\mathbf{x}_-)]\!] \right). \tag{2.42}$$

Since the $0/1$ loss is a non-differentiable function, to practically optimise the loss of the scoring function f on dataset $\mathcal{D}$, one approach is to upper bound the $0/1$ loss with one of its convex surrogates $\ell(z) \geq [\![z \leq 0]\!]$ (e.g., the exponential loss $\ell(z) = e^{-z}$, the logistic loss $\ell(z) = \log(1 + e^{-z})$, or the squared hinge loss $\ell(z) = [\max(0, 1 - z)]^2$). In other words, we can upper bound the misranking loss as

$$R^{\text{BR}}_{0/1}(f, \mathcal{D}) \leq \sum_{\mathbf{x}_+ \in \mathcal{S}_+} \sum_{\mathbf{x}_- \in \mathcal{S}_-} \ell(f(\mathbf{x}_+) - f(\mathbf{x}_-)).$$

If we measure the quality of a ranking function by the area under the ROC curve (AUC), loss functions for bipartite ranking are often variants of the misranking loss,

typical example is the loss function for logistic regression (i.e., the log loss or the cross entropy loss), where the prediction function $f : \mathbb{R}^D \to [0, 1]$ outputs a probability and the loss of f on $\mathcal{D}$ is $R^{\log}(f, \mathcal{D}) = -\sum_{\mathbf{x}_+ \in \mathcal{S}_+} \log f(\mathbf{x}_+) - \sum_{\mathbf{x}_- \in \mathcal{S}_-} \log(1 - f(\mathbf{x}_-))$.

which is related to $1-$ AUC (Ertekin and Rudin, 2011). For example, the objective of *P-Norm Push* is defined as (Rudin, 2009):

$$R_\ell^{\text{PN}}(f,\mathcal{D}) = \sum_{\mathbf{x}_-\in\mathcal{S}_-}\left[\sum_{\mathbf{x}_+\in\mathcal{S}_+}\ell(f(\mathbf{x}_+)-f(\mathbf{x}_-))\right]^p, \tag{2.43}$$

where $p\in\mathbb{R}^+$ is a parameter that acts as a soft maximum for the highest scoring negative example. It reduces to the objective of RankBoost if we employ the exponential surrogate loss $\ell(z)=e^{-z}$ and let $p=1$ (Freund et al., 2003).

Recalling the definition of infinity norm (or maximum norm)

$$\|\mathbf{z}\|_\infty = \max_i |z_i| = \lim_{p\to+\infty}\left(\sum_i |z_i|^p\right)^{1/p},$$

we have

$$R_\ell^{\infty}(f,\mathcal{D}) = \lim_{p\to+\infty}\left[R_\ell^{\text{PN}}(f,\mathcal{D})\right]^{\frac{1}{p}} = \max_{\mathbf{x}_-\in\mathcal{S}_-}\left[\sum_{\mathbf{x}_+\in\mathcal{S}_+}\ell(f(\mathbf{x}_+)-f(\mathbf{x}_-))\right], \tag{2.44}$$

which is the objective of *Infinite Push* (Agarwal, 2011). Further,

$$R_\ell^{\text{TP}}(f,\mathcal{D}) = \sum_{\mathbf{x}_+\in\mathcal{S}_+}\ell\left(f(\mathbf{x}_+)-\max_{\mathbf{x}_-\in\mathcal{S}_-}f(\mathbf{x}_-)\right), \tag{2.45}$$

which is the objective of *Top Push*, and $R_\ell^{\text{TP}}=R_\ell^{\infty}$ if the convex surrogate of the $0/1$ loss $\ell(\cdot)$ is non-increasing and differentiable (Li et al., 2014).

Intuitively, the Top Push penalises the scenario where any positive example is ranked below the highest-ranked negative example. If we penalise the scenario where any negative example is ranked above the lowest-ranked positive example, this results in the objective of *Bottom Push* (Rudin, 2009),

$$R_\ell^{\text{BP}}(f,\mathcal{D}) = \sum_{\mathbf{x}_-\in\mathcal{S}_-}\ell\left(\min_{\mathbf{x}_+\in\mathcal{S}_+}f(\mathbf{x}_+)-f(\mathbf{x}_-)\right). \tag{2.46}$$

It has been shown that the loss functions of binary classification and those of bipartite ranking are closely related (Ertekin and Rudin, 2011; Menon and Williamson, 2016). In particular, Ertekin and Rudin (2011) showed that the P-Norm Push and P-Classification share the same minimiser(s) when the scoring function takes the form of a linear function, and the P-Norm Push employs the exponential surrogate loss $\ell(z)=e^{-z}$. In addition, it also showed the RankBoost and Cost-Sensitive AdaBoost share the same minimiser(s). These results can be generalised to a parametric family of bipartite ranking and binary classification losses, see Appendix B for details.

2.6 Multi-task learning

The last technique we review is the multi-task learning paradigm, which will be employed to jointly learn user representations in Chapter 5. Given a number of related tasks, a natural approach is to learn each task independently. However, many work suggest that learning all the related tasks simultaneously can generally enjoy better generalisation performance (Caruana, 1993, 1997; Ruder, 2017). This paradigm of learning multiple tasks at the same time is called *multi-task learning*.

A typical approach to multi-task learning is to optimise the losses of all related tasks at the same time with shared representations, i.e., hard parameter sharing (Ruder, 2017). In particular, given T related tasks, we optimise the following objective:

$$\min_{\theta} \ \Omega(\theta) + \sum_{t=1}^{T} R_t(\theta), \tag{2.47}$$

where θ is the shared parameters, $\Omega(\theta)$ is the regularisation term, and $R_t(\theta)$ is the empirical risk of the t-th task.

An alternative approach to multi-task learning is to use separate representations for different tasks, but encourage related tasks to have similar representations, e.g., via regularisation, i.e., soft parameter sharing (Ruder, 2017). Suppose we use ℓ_2 regularisation to encourage similar parameters for related tasks, we can optimise

$$\min_{\theta_1,\ldots,\theta_T} \ \frac{\lambda}{2} \sum_{t,t'} \|\theta_t - \theta_{t'}\|^2 + \sum_{t=1}^{T} R_t(\theta_t), \tag{2.48}$$

where $\lambda \in \mathbb{R}^+$ is a regularisation constant, tasks t and t' are related, and θ_t is the parameter for task t.

A particular application of the multi-task learning paradigm is to perform *multi-label classification*, where we have a set of m class labels $\{l_1, \ldots, l_m\}$, and an instance $\mathbf{x} \in \mathbb{R}^D$ is associated with a subset of labels, which can be represented with a m dimensional binary vector $\mathbf{y} = (y_1, \ldots, y_m)$ where $y_j = 1$, $j \in \{1, \ldots, m\}$ if and only if label l_j is associated with the instance $\mathbf{x}$.

Formally, given a set of N instances and their associated labels $\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ as the training set, multi-label classification is to learn a function $f : \mathbb{R}^D \to \{0,1\}^m$ (from $\mathcal{S}$) that can predict the set of labels for any new instance $\mathbf{x}'$ not seen in $\mathcal{S}$. In practice, both the hard parameter sharing approach (Huang et al., 2013; Wang et al., 2016; Li et al., 2017) and the soft parameter sharing approach (Xue et al., 2007; Read et al., 2009; Dembczyński et al., 2010; Guo et al., 2011; Wu and Zhou, 2017) have been successfully employed to solve multi-label classification problems.

2.7 Summary

This chapter presents the problem of recommending structured objects (Section 2.1) and reviews the classic techniques for recommender systems (Section 2.2). We focus on the path and set recommendation problems and review techniques that enable efficiently training (Section 2.3) and inference (Section 2.4) for structured prediction, the loss functions of bipartite ranking and binary classification (Section 2.5), as well as the multi-task learning paradigm (Section 2.6). These techniques form the foundation of efficient recommendation of paths and sets. In the next chapter, we investigate the problem of recommending travel trajectories – an instance of path recommendation.

Chapter 3

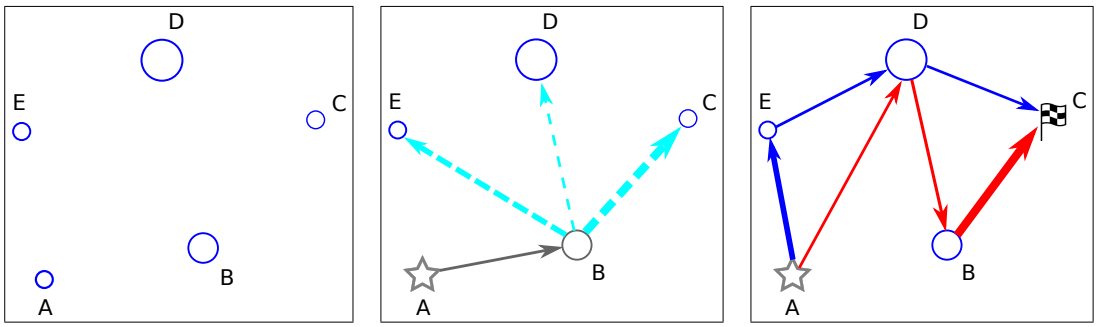
Feature-based Travel Trajectory Recommendation

In this chapter, we study the problem of recommending trajectories to travellers. A travel trajectory (or tour) consists of a sequence of points-of-interest (POIs). Further, a traveller is unlikely to make multiple visits to the same POI in a tour (in general), which suggests the recommended trajectory should be a *path* (Section 2.1.1). We describe the typical settings of travel recommendation in Section 3.1, and investigate a number of approaches to recommend paths based on POI ranking and route planning (Section 3.5), followed by empirical evaluation and discussion (Section 3.6).

3.1 Introduction

A large amount of location traces are becoming available from ubiquitous location tracking devices. For example, FourSquare has more than 50 million monthly users who have made 12 billion check-ins (Foursquare, 2019), and Flickr hosts over 5 billion photos, many with geolocation metadata (Flickr, 2019). This growing trend in rich geolocation data provides new opportunities for better travel planning traditionally done with written travel guides. Good solutions to these problems will in turn lead to better urban experiences for residents and visitors alike, and foster sharing of even more location-based behavioural data. This work proposes a novel solution to recommend travel routes in cities.

There are several typical settings of recommendation problems for locations and routes, as illustrated in Figure 3.1. The first setting can be called *POI recommendation* (Figure 3.1a). Each location (A to E) is scored with geographic and behavioural information such as category, reviews, popularity, spatial information such as distance, and temporal information such as travel time uncertainty, time of the day or day of the week. Figure 3.1b illustrates the second setting: *next location recommendation*, where the input is a partial trajectory (e.g., started at point A and currently at point B), the



(a) POI Recommendation (b) Next Location Recommendation (c) Trajectory Recommendation

Figure 3.1: Three settings of travel recommendation problems. Node size: POI score; edge width: transition score between pairs of POIs; grey: observed; star: starting location; flag: ending location. See Section 3.1 for details.

task of the algorithm is to score the next candidate location (e.g, C, D and E) based on the perceived POI score and transition compatibility with input $A \rightarrow B$. It is a variant of POI recommendation except both the user and locations travelled to date are given.

Here we consider the final setting: *trajectory recommendation* (Figure 3.1c), where the input are some factors about the desired route, e.g., starting point A and end point C, along with auxiliary information such as the desired length of trip. The algorithm needs to take into account location desirability (as indicated by node size) and transition compatibility (as indicated by edge width), and compare route hypotheses such as A-D-B-C and A-E-D-C.

All these settings have been broadly studied (Bao et al., 2015; Zheng, 2015; Zheng et al., 2014). We note, however, that two desired qualities are still missing from the current solutions to trajectory recommendation. The first is a principled method to jointly learn POI ranking (a prediction problem) and optimise for route creation (a planning problem). The second is a unified way to incorporate various features such as location, time, distance, user profile and social interactions, as they tend to get specialised and separate treatments. This work aims to address both challenges. Our main contributions are as follows:

- We propose a novel algorithm to jointly optimise point preferences and routes. We find that learning-based approaches generally outperform heuristic route recommendation (Lim et al., 2015). Incorporating transitions into POI ranking results in a better sequence of POIs, and avoiding sub-tours further improves performance of classical Markov chain methods.
- Our approach is feature-driven and learns from past behaviour without having to design specialised treatment for spatial, temporal or social information. It incorporates information about location, POI categories and behaviour history, and can use additional time, user, or social information if available.

- We show good performance compared to recent results (Lim et al., 2015), and also quantify the contributions from different components, such as ranking points, scoring transitions, and routing.
- Lastly, we propose a new metric, pairs-F_1, to evaluate trajectories, it captures the order in which POIs are visited. Pairs-F_1 lies between 0 and 1, and achieves 1 if and only if the recommended trajectory is exactly the same as the ground truth.

3.2 Problem statement

Recall that a travel trajectory or tour is a sequence of POIs. Given a set of POIs $\mathcal{P}$ in a city, and a set of historical travel trajectories from visitors, the *travel trajectory recommendation* problem is to suggest a trajectory through the city with regards to user specified constraints. In this thesis, we use a *trajectory query* to capture the constraints and group the tours that satisfy these constraints to form a training set $\mathcal{S} = \{(q^{(i)}, \{\mathbf{y}^{(ij)}\}_{j=1}^{N_i})\}_{i=1}^{N}$, where $q^{(i)}$ is the i-th query and $\mathbf{y}^{(ij)}$ is one of the N_i trajectories with respect to the constraints specified in $q^{(i)}$. Typically, no POI will be visited more than once in a tour (i.e., a tour is a *path*), and therefore we formulate the problem of recommending travel trajectories as an instance of path recommendation (Section 2.1.1). In this chapter, we consider the task of recommending one trajectory for a given query, the problem of recommending multiple trajectories will be addressed in Chapter 4. Further, we define a trajectory query as $q = (p_s, p_e, L)$, where p_s, $p_e \in \mathcal{P}$ are respectively the start and end locations of a desired trajectory, and L is the number of POIs (including p_s and p_e) the user wish to visit.

There have been other formulations of the tour recommendation problem (Gionis et al., 2014; Lim et al., 2015; Chen et al., 2017b; He et al., 2018), for example, instead of constraining the number of POIs to visit, Lim et al. (2015) and He et al. (2018) used a query $q = (p_s, p_e, T)$ where T is the time budget for a desired trajectory; Gionis et al. (2014), on the other hand, restricted the maximum distance of a trajectory, i.e., with a query $q = (p_s, p_e, D)$ where a recommended trajectory can not have a distance larger than D. There is also work that omits the end location from the query to accommodate other recommendation scenarios, which results in a query $q = (p_s, L)$ (Chen et al., 2017b). Further, the query can be generalised to include constraints such as the venue type, or any restrictions that might be specified by the end user (Gionis et al., 2014).

3.3 Related work

We summarise recent work most related to formulating and solving learning problems on assembling routes from POIs, and refer the reader to recent surveys (Bao et al., 2015; Zheng, 2015; Zheng et al., 2014) for general overviews of the area.

3.3.1 Solutions for typical travel recommendation problems

A popular approach for the *POI recommendation* problem (Figure 3.1a) is to recommend POIs with a collaborative filtering model that learns user-location affinity (Shi et al., 2011), with additional ways to incorporate spatial (Lian et al., 2014; Liu et al., 2014), temporal (Yuan et al., 2013; Hsieh and Li, 2014; Gao et al., 2013), or spatial-temporal (Yuan et al., 2014) information. It can also be in discovery mode, such as identifying points-of-interest (Zheng et al., 2009; Li et al., 2015) and includes efficient querying of geographic objects for trips (Hashem et al., 2015).

Current solutions to the *next location recommendation* problem (Figure 3.1b) include incorporating Markov chains into collaborative filtering (Rendle et al., 2010; Cheng et al., 2013; Zhang and Wang, 2015), quantifying tourist traffic flow between points-of-interest (Zheng et al., 2012), incorporating regional influence of POIs via latent factors (Lu et al., 2018), formulating a binary decision or ranking problem (Baraglia et al., 2013), and predicting the next location with sequence models such as recurrent neural networks (Liu et al., 2016b; Lu et al., 2018).

Existing work on *trajectory recommendation* (Figure 3.1c) either uses heuristic combination of locations and routes (Lu et al., 2010; Lim et al., 2015; Lu et al., 2012), or formulates an optimisation problem that is not informed or evaluated by behaviour history (Gionis et al., 2014; Chen et al., 2015). Joint learning of location preferences and routes remains an open problem. Motivated by this observation, we formulate a learning problem to score the whole trajectory by taking into account both individual POI properties and relationships among different POIs.

3.3.2 Methods for ranking locations and trajectories

The classic collaborative filtering, or the matrix factorisation family of techniques have been applied to places and trajectories to learn the hidden representations of POI and trajectory (Shi et al., 2011; Cheng et al., 2013; Zhang and Wang, 2015), which can then be employed to rank POIs and trajectories. Alternatively, user specific scores for locations can be assigned according to, e.g., user preference and popularity of POIs (Debnath et al., 2018). Another type of method regards route recommendation as a planning problem (Gionis et al., 2014; Lim et al., 2015; He et al., 2018). The TripBuilder system (Brilhante et al., 2013) first solves a maximum coverage problem for user preference and then solves the TSP for routing. He et al. (2018) suggest trajectories using integer programming and local search that leverage user and POI embeddings jointly learned by the Bayesian Personalized Ranking approach. Anagnostopoulos et al. (2017) recommend tours for a group of travellers by solving a planning problem that generalises the orienteering problem. Chen et al. (2018) adopt the Q-learning

algorithm to successively generate personalised bicycle trips.

The collaborative filtering approaches rank POIs but do not take into account the sequence that a trip is taken. On the other hand, the planning approaches assume a fixed objective function that is not directly optimised to predict user behaviour. There have been a few approaches that jointly consider POI preferences and routes (Lu et al., 2012; Kurashima et al., 2010; Chen et al., 2015).

3.3.3 Features and information employed

A diverse set of available information have been employed, such as geographic, time, user properties, and subjective opinions. Approaches for modelling space include geolocation-informed matrix factorisation (Lian et al., 2014), spatial topic models (Hu and Ester, 2013), neighbourhood information (Liu et al., 2014), exploiting sequential information with additive Markov chains (Zhang et al., 2014), and enriching location with the information for venues (Deveaud et al., 2014, 2015). Important variants to consider for modelling time include travel time (Gao et al., 2013), constructing time-aware routes (Yuan et al., 2013; Hsieh and Li, 2014), time-of-day and day-of-week (Chen et al., 2015), POI availability and uncertainty in travelling time (Zhang et al., 2015a). There are a number of approaches that jointly consider space and time (Yuan et al., 2014; Zhang and Wang, 2015), such as modelling the correlation between check-in time and location (Gao et al., 2013).

Inferring user attributes and preferences has been an important consideration (Liu et al., 2013). Chen et al. (2013) recommend places to travellers based on user demographics and travel group types (e.g., couple, family and friends) from online photos. Ference et al. (2013) tailors the recommendation for out of town users. Subjective opinion is an important information source for decision making, in addition to past behaviours. Zhang et al. (2015b) use written reviews for POI recommendation. Le and Pishva (2016) integrate the weather condition, public transportation and cultural events in a number of Japanese cities to form tours that account for users' time and budget constraints. The novel work from Quercia et al. (2014) crowd-source judgements about the beauty, quietness and happiness of places, and then constructs routes with these subjective criteria.

3.4 Query features and POI transition

In the tour recommendation problem, the training data consists of a set of tours of varying length in a particular city. We consider only POIs that have been visited by at least one user in the past, and construct a graph with POIs as nodes and directed

edges representing the observed transitions between pairs of POIs in tours.

We extract the category, popularity (number of distinct visitors) (De Choudhury et al., 2010), total number of visits and average visit duration for each POI. POIs are grouped into 5 clusters using K-means according to their geographical locations to reflect their neighbourhood. Furthermore, since we are constrained by the fact that trajectories have to be of length L and start and end at certain points, we hope to improve the recommendation by using this information. In other words, we use the query $q = (p_s, p_e, L)$ to construct new features by contrasting candidate POIs with p_s and p_e. For each of the POI features (i.e., category, neighbourhood, popularity, total visits and average duration), we construct two new features by taking the difference of the feature in POI p with p_s and p_e respectively. For the category (or neighbourhood), we set the feature to 1 when their categories (or cluster identities) are the same and -1 otherwise. For popularity, total visits and average duration, we take the real valued difference. Lastly, we compute the distance from POI p to p_s (and p_e) using the Haversine formula (Sinnott, 1984), and also include the required length L as a feature.

In addition to information about each individual POI, a tour recommendation system would benefit from capturing the likelihood of going from one POI to another different POI. One option would be to directly model the probability of going from any POI to any other POI, but this has several weaknesses: Such a model would be unable to handle a new POI (one that has not yet been visited), or pairs of existing POIs that do not have an observed transition. Furthermore, even if we restrict ourselves to known POIs and transitions, there may be locations which are rarely visited, leading to significant challenges in estimating the probabilities from empirical data.

We model POI transitions using a Markov chain with discrete states by factorising the transition probability $\mathbb{P}(p_j|p_i)$, $i, j \in \{1, \ldots, |\mathcal{P}|\}$ as a product of transition probabilities between pairs of individual POI features, assuming independence between these feature-wise transitions. The popularity, total visits and average duration are discretised by binning them uniformly into 5 intervals on the log scale. These feature-to-feature transitions are estimated from data using maximum likelihood principle. The POI-POI transition probabilities can be efficiently computed by taking the Kronecker product of transition matrices for the individual features, and then updating it based on appropriate normalisation as well as three additional constraints. First we disallow self-loops by setting the probability of $\mathbb{P}(p_i|p_i)$, $i \in \{1, \ldots, |\mathcal{P}|\}$ to zero. Secondly, when a group of multiple POIs have identical (discretised) features, we distribute the transition probability uniformly among POIs in the group. In particular, the incoming transition probability (from other POIs or the group itself) is distributed uniformly among POIs in the group, while taking care of the requirement that no POI self-loop is allowed. The outgoing transition probability of a POI in the group is set

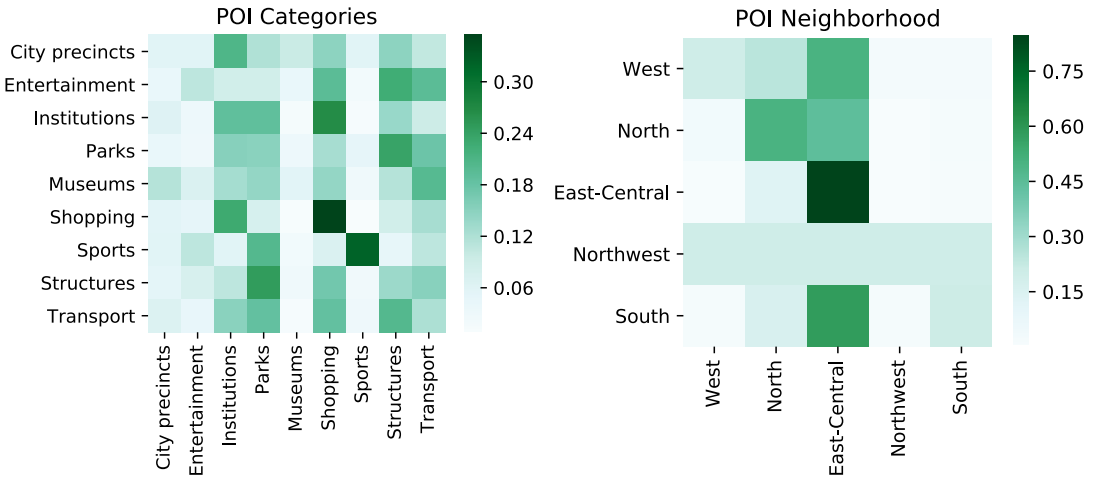


Figure 3.2: Transition matrices for two POI features from Melbourne: POI category and neighbourhood.

to equal that of the group. Third, we remove feature combinations that has no POI in dataset. Figure 3.2 visualises the transition matrices for two POI features, category and neighbourhood, in Melbourne.

3.5 Tour recommendation

In this section, we first describe methods to recommend points and routes, then propose an approach to combine them for trajectory recommendation. Lastly, we propose a method to avoid sub-tours and describe alternative approaches briefly.

3.5.1 POI ranking and route planning

A naive approach would be to recommend trajectories based on the popularity of POIs only, that is we always suggest the top-k most popular POIs for all visitors given the start and end location. We call this baseline approach PoiPopularity, and its only adaptation to a particular query is to adjust k to match the desired length.

On the other hand, we can leverage the whole set of POI-query features (Section 3.4) to learn a ranking of POIs (with respect to a query). Suppose we use the rankSVM (Lee and Lin, 2014) with a linear kernel and the squared hinge loss,

$$\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^\top \mathbf{w} + C \sum_{q \in \mathcal{Q},\ y_{iq} > y_{jq}} \max\left(0,\ 1 - \mathbf{w}^\top (\phi_{iq} - \phi_{jq})\right)^2, \tag{3.1}$$

where $\mathbf{w}$ is a parameter vector, $C \in \mathbb{R}^+$ is a regularisation constant, $\mathcal{Q}$ denotes the set of queries from trajectories in the training set, and y_{iq}, $i \in \{1, \ldots, |\mathcal{P}|\}$, $q \in \mathcal{Q}$ is

the label of the POI-query pair (p_i, q), which is the number of occurrences of POI p_i in the set of training trajectories corresponding to query q, without counting the occurrence of p_i when it is the origin or destination of a trajectory. Lastly, ϕ_{iq} is a vector of POI-query features of (p_i, q) as described in Section 3.4.

The ranking score of p_i with respect to query q is computed as $R_{iq} = \mathbf{w}^\top \phi_{iq}$. Intuitively, we learn a function that scores the affinity between POI p and query q, however, only the rank (instead of the real-valued affinity score) of the POI with respect to the given query matters.

Given a query $q = (p_s, p_e, L)$, we can recommend a trajectory by first ranking the set of POIs $\mathcal{P} \setminus \{p_s, p_e\}$ (with respect to query q), then take the top ranked $L-2$ POIs and connect them according to the ranks. We call this method PoiRank.

In addition to recommending trajectories by ranking POIs, we can leverage the POI-POI transition probabilities and recommend a trajectory for a particular query by maximising the transition likelihood. The maximum likelihood of the Markov chain of transitions can be found using a variant of the Viterbi algorithm (with uniform emission probabilities). We refer to this approach that only uses the transition probabilities between POIs as Markov.

3.5.2 Combining ranking and transition

We would like to leverage both point ranking and transitions, i.e., recommending a trajectory that maximises the points ranking of its POIs as well as its transition likelihood at the same time.

One option to find a trajectory that simultaneously maximises the ranking probabilities of its POIs and its transition likelihood is to optimise an objective that accounts for both POI ranking and transition likelihood. Here we consider an instance of this objective, it aggregates the contributions of POIs ranking and transitions by trading-off the importance between the two components using a parameter $\alpha \in [0, 1]$. In practice, the value of parameter α is generally tuned using cross validation.

In other words, we optimise the following objective:

$$\underset{\mathbf{y} \in \mathcal{P}^L}{\operatorname{argmax}} \ \alpha \sum_{k=2}^{L-1} \log \mathbb{P}_R(p_k|q) + (1-\alpha) \sum_{k=1}^{L-1} \log \mathbb{P}(p_{k+1}|p_k), \tag{3.2}$$

where $\mathbf{y} = (p_1, \ldots, p_L)$ is any possible trajectory, and we transform the ranking scores of POIs with respect to a query to a probability distribution using the softmax function,

$$\mathbb{P}_R(p_i|q) = \frac{\exp(R_{iq})}{\sum_{p_j \in \mathcal{P} \setminus \{p_s, p_e\}} \exp(R_{jq})}, \ p_i \in \mathcal{P} \setminus \{p_s, p_e\}. \tag{3.3}$$

Algorithm 4 RANK+MARKOV: recommend trajectory with POI ranking and transition

Input: $\mathcal{P}, p_s, p_e, L$
Output: Trajectory $\mathbf{y} = (p_s, \cdots, p_e)$ with L POIs
Initialise score matrix A and backtracking-point matrix B
for $p \in \mathcal{P}$ **do**
  $A[2, p] = S(p; p_s, q)$, $B[2, p] = p_s$
end for
for $l = 2$ to $L - 1$ **do**
  for $p \in \mathcal{P}$ **do**
    $A[l+1, p] = \max_{p' \in \mathcal{P}} \{A[l, p'] + S(p; p', q)\}$
    $B[l+1, p] = \text{argmax}_{p' \in \mathcal{P}} \{A[l, p'] + S(p; p', q)\}$
  end for
end for
$\mathbf{y} = (p_e)$, $l = L$, $p = p_e$
repeat
  Prepend $B[l, p]$ to $\mathbf{y}$
  $l = l - 1$, $p = B[l, p]$
until $l < 2$
return $\mathbf{y}$

To maximise the objective in (3.2), let $S(p; p', q)$ be a convex combination of point ranking and transition (from p' to p),

$$S(p; p', q) = \alpha \log \mathbb{P}_R(p|q) + (1 - \alpha) \log \mathbb{P}(p|p'), \tag{3.4}$$

we can then find the best path (or walk) using the Viterbi algorithm. We call this approach that uses both the point ranking and transitions RANK+MARKOV.

Algorithm 4 shows the pseudo code of this method, where A is the score matrix, and entry $A[l, p]$ stores the maximum value associated with the (partial) trajectory that starts at p_s and ends at p with l POI visits; B is the backtracking-point matrix, and entry $B[l, p]$ stores the predecessor of p in that (partial) trajectory. The maximum objective value can be found in $A[L, p_e]$, and the corresponding trajectory can be built by tracing back from $B[L, p_e]$.

3.5.3 Avoiding sub-tours

Trajectories recommended by MARKOV (Section 3.5.1) and RANK+MARKOV (Section 3.5.2) are found using the maximum likelihood approach, and may contain multiple visits to the same POI. This is because the best solution from Viterbi decoding may have circular sub-tours (where a POI already visited earlier in the tour is visited again). We propose a method for eliminating sub-tours by finding the best path using an

integer linear program (ILP), with sub-tour elimination constraints adapted from the ILP formulation of the s-t path TSP (Section 2.4.3). In particular, we recommend a trajectory for query $q = (p_s, p_e, L)$ by solving the ILP with objective (3.5) and constraints (3.6)-(3.10), where $M = |\mathcal{P}|$ is the number of available POIs and binary variable z_{ij} determines whether the transition from p_i to p_j is in the resulting trajectory.

$$\max_{\mathbf{z},\mathbf{u}} \sum_{i=1}^{M-1} \sum_{j=2}^{M} z_{ij} \log \mathbb{P}(p_j|p_i) \tag{3.5}$$

$$s.t. \quad z_{ij} \in \{0,1\},\ z_{ii} = 0,\ u_i \in \mathbb{Z},\ \forall i,j \in \{1,\cdots,M\} \tag{3.6}$$

$$\sum_{j=2}^{M} z_{1j} = \sum_{i=1}^{M-1} z_{iM} = 1,\ \sum_{i=2}^{M} z_{i1} = \sum_{j=1}^{M-1} z_{Mj} = 0 \tag{3.7}$$

$$\sum_{i=1}^{M-1} z_{ik} = \sum_{j=2}^{M} z_{kj} \leq 1,\ \forall k \in \{2,\cdots,M-1\} \tag{3.8}$$

$$\sum_{i=1}^{M-1} \sum_{j=2}^{M} z_{ij} = L - 1, \tag{3.9}$$

$$u_i - u_j + 1 \leq (M-1)(1 - z_{ij}),\ \forall i,j \in \{2,\cdots,M\} \tag{3.10}$$

For brevity, we arrange the POIs such that $p_1 = p_s$ and $p_M = p_e$. Firstly, the desired trajectory should start from p_s and end at p_e (Constraint 3.7). In addition, any POI could be visited at most once (Constraint 3.8), and only $L-1$ transitions between POIs are permitted (Constraint 3.9), i.e., the number of POI visits should be exactly L (including both p_s and p_e). The last constraint (3.10), where u_i is an auxiliary variable, enforces that only a single sequence of POIs without sub-tours is permitted in the trajectory (Miller et al., 1960).

We can solve this ILP using off-the-shelf ILP solvers such as the CPLEX optimiser[1] or the Gurobi optimisation package[2]. The resulting trajectory can be constructed by tracing the non-zeros in binary variables $\mathbf{x}$. We call the method that uses the POI-POI transition matrix to recommend paths without circular sub-tours MarkovPath.

Note that the sub-tours in trajectories recommended by the Rank+Markov method can be eliminated in a similar manner. In particular, we can retain the constraints (3.6)-(3.10) but modify the objective of the ILP as

$$\max_{\mathbf{z},\mathbf{u}} \sum_{i=1}^{M-1} \sum_{j=2}^{M} z_{ij}\, S(p_j; p_i, q), \tag{3.11}$$

where $S(p_j; p_i, q)$ is defined in Equation (3.4). In the experiments, this method is referred to as Rank+MarkovPath.

[1] https://www.ibm.com/analytics/cplex-optimizer
[2] http://www.gurobi.com

3.5.4 Incorporating time constraints

The ILP formulation (3.5)-(3.10) can be adapted to deal with time constraints (e.g., time budget for a desired trajectory, time-of-the-day constraints on POIs), which could sometimes be more readily available than the number of POIs to be visited. In particular, let c_i denote the time to be spent at POI p_i and t_{ij} be the time to travel from p_i to p_j, and suppose both c_i and t_{ij}, $i, j \in \{1, \ldots, M\}$ are available before we make recommendations. Further, if we have a query with time constraints $q = (p_s, p_e, T)$ where T is the time budget for a desired trajectory, we can create a constraint that incorporates these information:

$$\sum_{i=1}^{M-1} \sum_{j=2}^{M} z_{ij} \left(t_{ij} + \frac{1}{2}(c_i + c_j) \right) + \frac{1}{2}(c_1 + c_M) \leq T, \tag{3.12}$$

where the scaling factor $1/2$ prevents the time to be spent at POIs from being counted twice. Constraint (3.12) restricts that the total time to be spent in a desired trajectory should not exceed the specified time budget.

Note that optimising objective (3.5) by simply replacing the trip length constraint (3.9) with (3.12) could result in trivial recommendations (i.e., suggesting the user to visit only the start and end locations specified in query), this is because the (log) transition likelihood of a trajectory that objective (3.5) maximises generally becomes smaller as the trajectory gets longer, which forces the ILP to prefer the shortest trajectory that satisfies all specified constraints, i.e., a trajectory with the (given) start and end POIs only. One approach to avoid such trivial recommendations is restricting the least amount of time that shall be spent, for example, via the following constraint

$$\gamma T \leq \sum_{i=1}^{M-1} \sum_{j=2}^{M} z_{ij} \left(t_{ij} + \frac{1}{2}(c_i + c_j) \right) + \frac{1}{2}(c_1 + c_M) \leq T, \tag{3.13}$$

where $0 \leq \gamma \leq 1$ is a parameter that controls the lower bound of time actually spent in the time budget. We therefore create two additional methods that optimise objectives (3.5) and (3.11) by replacing constraint (3.9) with (3.13). We call these methods MarkovPath-T and Rank+MarkovPath-T respectively in experiment.

We remark that one can also employ both the trip length constraint and the time constraint in the above approach to further restrict a desired trajectory. In addition, this approach can be extended to incorporate other time-related constraints, e.g., time-of-the-day constraints on POIs (i.e., POIs might be only available in a limited time frame in a day), and we present such a method in Appendix C.

3.5.5 Discussion

Instead of ranking POIs and modelling transition patterns separately, one can directly learn sequence models from historical travel trajectories in a city. In particular, struc-

tured prediction methods (Section 2.3) such as structured support vector machines (SSVMs) (Tsochantaridis et al., 2004; Taskar et al., 2004) and conditional random fields (CRFs) (Lafferty et al., 2001), have been widely adopted for such tasks (Joachims et al., 2009b; Tsochantaridis et al., 2005; Wallach, 2004; Sutton et al., 2012). In Chapter 4, we explore this approach for path recommendation.

In recent years, recurrent neural networks (RNNs) and one dimensional convolutional neural networks (1D-CNNs) that can capture long-term dependences or spatial patterns in sequences have been successfully applied to sequential data (Abdel-Hamid et al., 2014; Liu et al., 2016a,b; Yang et al., 2017), it is likely that these methods can also be effective for the task of tour recommendation. However, we want to remark that the outputs of these approaches are not naturally *paths*, therefore the inference procedures may need extra processing.

3.5.6 Measuring performance

A commonly used metric for evaluating the recommended POIs and trajectories is the F_1 score on points, which is the harmonic mean of precision and recall of the set of POIs in a recommended trajectory versus that in the observed ground truth (Lim et al., 2015). While being good at measuring how well the set of observed POIs are correctly recommended, F_1 score on points ignores the visiting order between POIs.

We propose a new metric, *pairs-F_1* (or F_1 score on pairs), that considers both the POI identity and the visiting order in a trajectory. It measures the F_1 score of (the set of) all possible pairs of POIs (whether they are adjacent or not) in a recommended trajectory versus those in the observed tour,

$$\text{pairs-F}_1 = \frac{2P_{\text{pair}}R_{\text{pair}}}{P_{\text{pair}} + R_{\text{pair}}},$$

where P_{pair} and R_{pair} are the precision and recall of ordered POI pairs, respectively:

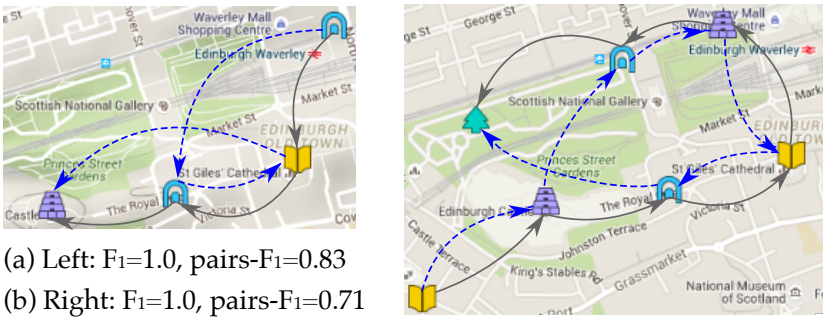
$$P_{\text{pair}} = \frac{N_c}{|\mathcal{P}_{\hat{\mathbf{y}}}|(|\mathcal{P}_{\hat{\mathbf{y}}}|-1)/2}, \; R_{\text{pair}} = \frac{N_c}{|\mathcal{P}_{\mathbf{y}}|(|\mathcal{P}_{\mathbf{y}}|-1)/2},$$

here $\hat{\mathbf{y}}$ is the recommended trajectory and $\mathbf{y}$ is the observed ground truth, $\mathcal{P}_{\mathbf{y}}$ denotes the set of POIs in trajectory $\mathbf{y}$, and N_c is the number of ordered POI pairs[3] (p_i, p_j) that appear in both the ground-truth and the recommended trajectory, i.e.,

$$N_c = \left|\left\{(p_i, p_j) : p_i \prec_{\mathbf{y}} p_j,\ p_i, p_j \in \mathcal{P}_{\mathbf{y}}\right\} \cap \left\{(p_i, p_j) : p_i \prec_{\hat{\mathbf{y}}} p_j,\ p_i, p_j \in \mathcal{P}_{\hat{\mathbf{y}}}\right\}\right|,$$

where $p_i \prec_{\mathbf{y}} p_j$ denotes that POI p_i was visited before p_j in trajectory $\mathbf{y}$.

[3] We define pairs-$F_1 = 0$ when $N_c = 0$.



(a) Left: F_1=1.0, pairs-F_1=0.83
(b) Right: F_1=1.0, pairs-F_1=0.71

Figure 3.3: Examples of F_1 versus pairs-F_1 as evaluation metric. Solid grey: ground truths; dashed blue: recommended trajectories. See Section 3.5.6 for details.

Pairs-F_1 takes values between 0 and 1 (higher is better). A perfect pairs-F_1 is achieved *if and only if* both the POIs and their visiting order in the recommended trajectory are exactly the same as those in the ground truth trajectory. On the other hand, pairs-F_1 equals 0 means none of the recommended POI pairs was actually visited (in the designated order) in the real trajectory.

Figure 3.3 illustrates the difference between F_1 score on points and F_1 score on pairs, where the solid grey lines represent the ground truth transitions that actually visited by travellers, and the dashed blue lines are the transitions in a recommended trajectory. Both examples have a perfect F_1 score, but not a perfect pairs-F_1 score due to the difference in POI sequencing.

3.6 Experiments

In this section, we empirically evaluate the proposed path recommendation approaches for recommending travel trajectories, and compare them with a number of baselines as well as cutting-edge methods.

3.6.1 Photo trajectories from five cities

We experiment on trajectory datasets from five cities, namely, Edinburgh, Glasgow, Osaka, Toronto and Melbourne. The first four datasets are provided by Lim et al. (2015) and the Melbourne dataset is built using the same approach introduced in earlier work (De Choudhury et al., 2010; Lim et al., 2015) and briefly described below.

Trajectories are extracted using geo-tagged photos in the Yahoo! Flickr Creative Commons 100M (YFCC100M) dataset (Thomee et al., 2016) as well as the Wikipedia web-pages of corresponding points-of-interest. Photos are mapped to POIs according to their distances calculated using the Haversine formula (Sinnott, 1984), the time a user arrived at a POI is approximated by the time the first photo taken by the user at

Table 3.1: Statistics of trajectory datasets.

Dataset	Photos	Visits	Trajectories	Users	POIs
Edinburgh	82,060	33,944	5,028	1,454	28
Glasgow	29,019	11,434	2,227	601	27
Osaka	392,420	7,747	1,115	450	27
Toronto	157,505	39,419	6,057	1,395	29
Melbourne	94,142	23,995	5,106	1,000	85

Table 3.2: Information captured by different trajectory recommendation methods.

Method	Query	POI	Transition	No sub-tours
Random	×	×	×	×
PersTour	×	√	×	√
PersTour-L	×	√	×	√
PoiPopularity	×	√	×	×
PoiRank	√	√	×	×
Markov	×	√	√	×
MarkovPath	×	√	√	√
MarkovPath-T	×	√	√	√
Rank+Markov	√	√	√	×
Rank+MarkovPath	√	√	√	√
Rank+MarkovPath-T	√	√	√	√

that POI, similarly, the time a user left a POI is approximated by the time the last photo taken by the user at that POI. Further, the sequence of POI visits by a particular user are divided into several segments according to the time gap (e.g., 8 hours) between consecutive POI visits, and the POI visits in each segment are connected in temporal order to form a trajectory. Table 3.1 summaries the datasets used in the experiments.

3.6.2 Experimental setup

We use leave-one-out cross validation to evaluate different trajectory recommendation methods, i.e., when testing on a trajectory, all other trajectories are used for training. We compare the trajectory recommendation methods described in Section 3.5 with a number of baselines from recent literature, these recommendation approaches are summarised below, with an overview in Table 3.2.

Baselines The Random method naively chooses POIs uniformly at random (without replacement) from the set of POIs $\mathcal{P} \setminus \{p_s, p_e\}$ to form a trajectory. It does not utilise any features related to POI or query, as shown in Table 3.2. Among the related

approaches, PersTour (Lim et al., 2015) is the most similar work and it explores both POI features and the sub-tour elimination constraints described in Section 3.5.3, in addition, the recommended trajectory is constrained by a time budget. A variant of PersTour that applies the trajectory length instead of the time budget to constrain the recommended trajectory, which is denoted as PersTour-L in Table 3.2. Another baseline is PoiPopularity described at the beginning of Section 3.5; it only employs POI popularity when recommending a trajectory.

Variants of point- and route-ranking approaches The PoiRank method presented in Section 3.5.1 employs both POI and query features; Markov (Section 3.5.1) recommends trajectories uses only transition probabilities between POIs, and its variant MarkovPath incorporates additional constraints to eliminate sub-tours. Both the Rank+Markov and Rank+MarkovPath methods (Section 3.5.2) utilise POI and query features as well as transition probabilities between POIs, with the latter approach exploits additional sub-tour elimination constraints. Lastly, we have MarkovPath-T and Rank+MarkovPath-T (Section 3.5.4) that replace the trip length constraint with time constraints in their corresponding variants MarkovPath and Rank+MarkovPath. We remove the trip length from the POI-query features, and use the average visit duration at POI p_i as a proxy for c_i, i.e., the time to be spent at p_i. In addition, we follow (Lim et al., 2015) to use a speed of 4 km per hour (i.e., walking) when computing the time needed to travel between POIs.

Hyper-parameters The trade-off parameter for PersTour and PersTour-L used in experiments is 0.5, which was found to be the best weighting in Lim et al. (2015). We set the regularisation constant in rankSVM (used in PoiRank, Rank+Markov and Rank+MarkovPath) to 10.0, which is determined by cross-validation. The trade-off parameter (i.e., α in Equation 3.4) for both Rank+Markov and Rank+MarkovPath is set to 0.5 (tuned by cross-validation). We use $\gamma = 0.5$ in the time constraint (3.13) for both MarkovPath-T and Rank+MarkovPath-T.

3.6.3 Results

Table 3.3 and Table 3.4 summarise the performance of various trajectory recommendation approaches, in terms of F_1 and pairs-F_1 scores, respectively. We remark the performance values consist of the *mean* and *standard error* of the corresponding metric.

It is unsurprising that methods which capture (some type of) information about the recommendation problem (as summarised in Table 3.2) outperform the Random baseline in terms of both metrics on all five datasets.

Table 3.3: Performance comparison on five datasets in terms of F_1 scores. The **best** method for each dataset is shown in bold, the *second best* is shown in italic.

Method	Edinburgh	Glasgow	Melbourne	Osaka	Toronto
Random	.570 ± .006	.632 ± .012	.558 ± .008	.621 ± .017	.621 ± .007
PersTour	.656 ± .009	**.801** ± .020	.483 ± .010	.686 ± .034	.720 ± .012
PersTour-L	.651 ± .006	.660 ± .010	.576 ± .007	.686 ± .020	.643 ± .006
PoiPopularity	**.701** ± .006	.745 ± .016	.620 ± .007	.663 ± .018	.678 ± .007
PoiRank	*.700* ± .006	*.768* ± .016	*.637* ± .007	**.745** ± .025	**.754** ± .009
Markov	.645 ± .007	.725 ± .016	.577 ± .008	.697 ± .022	.669 ± .008
MarkovPath	.678 ± .006	.732 ± .016	.595 ± .007	.706 ± .022	.688 ± .008
MarkovPath-T	.677 ± .006	.691 ± .013	.615 ± .007	.702 ± .019	.673 ± .008
Rank+Markov	.659 ± .007	.754 ± .016	.613 ± .008	.715 ± .024	.723 ± .010
Rank+MarkovPath	.697 ± .006	.762 ± .016	**.639** ± .007	*.732* ± .024	*.751* ± .009
Rank+MarkovPath-T	.671 ± .006	.683 ± .013	.625 ± .007	.700 ± .021	.680 ± .008

Table 3.4: Performance comparison on five datasets in terms of pairs-F_1 scores. The **best** method for each dataset is shown in bold, the *second best* is shown in italic.

Method	Edinburgh	Glasgow	Melbourne	Osaka	Toronto
Random	.261 ± .006	.320 ± .016	.248 ± .007	.304 ± .021	.310 ± .009
PersTour	.417 ± .014	**.643** ± .035	.216 ± .013	.468 ± .055	.504 ± .019
PersTour-L	.359 ± .008	.352 ± .015	.266 ± .007	.406 ± .035	.333 ± .009
PoiPopularity	*.436* ± .010	.507 ± .028	.316 ± .009	.365 ± .028	.384 ± .011
PoiRank	.432 ± .010	*.548* ± .030	.339 ± .010	**.511** ± .045	**.518** ± .016
Markov	.417 ± .010	.495 ± .028	.288 ± .010	.445 ± .039	.407 ± .013
MarkovPath	.400 ± .009	.485 ± .028	.294 ± .009	.442 ± .038	.405 ± .013
MarkovPath-T	.366 ± .007	.388 ± .019	.287 ± .008	.410 ± .032	.363 ± .011
Rank+Markov	**.444** ± .010	.545 ± .029	**.351** ± .011	.486 ± .042	.512 ± .017
Rank+MarkovPath	.428 ± .010	.533 ± .029	*.344* ± .010	*.489* ± .042	*.514* ± .016
Rank+MarkovPath-T	.358 ± .007	.369 ± .017	.297 ± .008	.407 ± .035	.368 ± .010

Is ranking POIs with respect to query helpful? We can see that methods based on ranking POIs (i.e., PoiRank, Rank+Markov, Rank+MarkovPath) yield strong performance in terms of both metrics. In particular, PoiRank improves notably upon PoiPopularity and PersTour by leveraging POI-query features. Further, methods incorporating POI ranking information (i.e., Rank+Markov and Rank+MarkovPath) always perform better than their respective counterparts with transition information alone (i.e., Markov and MarkovPath). This suggest that ranking POIs with respect to query is helpful for recommending trajectories.

Is incorporating transitions between POIs helpful? Table 3.3 shows that MARKOV which leverages only POI transitions does not perform particularly well in terms of F_1 score on points. However, we can see from Table 3.4 that most Markov chain entries have better performance in terms of F_1 score on pairs, specifically, the performance of RANK+MARKOV is either comparable or better than that of POIRANK across the five datasets, which indicates the Markov chain approaches generally respect the transition patterns between POIs.

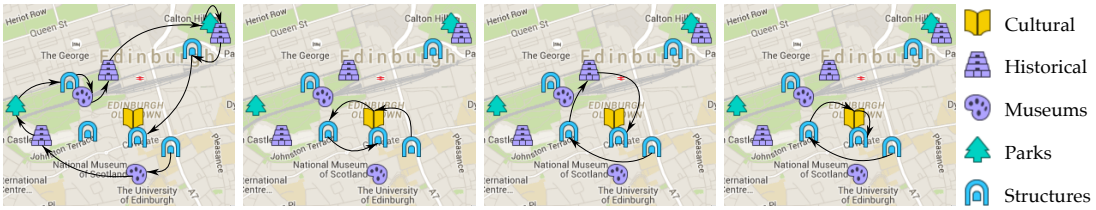
Is avoiding sub-tours important? We can see from Table 3.3 that MARKOVPATH and RANK+MARKOVPATH outperform their corresponding variants without the path constraints in terms of F_1 score on points, which demonstrates that eliminating sub-tours improves point recommendation. This is not unexpected, since sub-tours worsen the proportion of correctly recommended POIs as a length constraint is imposed.

Is incorporating time constraints helpful? PERSTOUR (Lim et al., 2015) always performs better than its variant PERSTOUR-L, in terms of both metrics, especially on `Glasgow` and `Toronto` datasets. This indicates the time budget constraint is more helpful than length constraint for recommending trajectories. Surprisingly, we observed that PERSTOUR is outperformed by RANDOM baseline on `Melbourne` dataset. It turns out that on this dataset, many of the ILP problems which PERSTOUR needs to solve to get the recommendations are difficult ILP instances. In the leave-one-out evaluation, although we employed a large scale computing cluster with modern hardware, 12% of evaluations failed as the ILP solver was unable to find a feasible solution after 2 hours. In addition, many recommendations were suboptimal solutions of the corresponding ILPs due to the time limit. These factors lead to the inconsistent performance of PERSTOUR on the `Melbourne` dataset.

In contrast, the performance of MARKOVPATH-T is comparable to that of MARKOVPATH in terms of F_1 score on points (Table 3.3), except slight variations on the `Glasgow` and `Melbourne` datasets. However, MARKOVPATH-T performs consistently worse than MARKOVPATH in terms of F_1 score on pairs (Table 3.4). On the other hand, RANK+MARKOVPATH outperforms RANK+MARKOVPATH-T in terms of both metrics on all five datasets. These interesting results might suggest that time constraints are not always more helpful than constraining the trip length for travel trajectory recommendation, and further investigation is likely required for a better understanding.

3.6.4 An illustrative example

Figure 3.4 illustrates an example from Edinburgh. The ground truth is a trajectory of length 4 that starts at a POI of category *Structures*, visits two intermediate POIs of category *Structures* and *Cultural* and terminates at a POI of category *Structures*.



(a) PersTour (b) PoiRank (c) Markov (d) Best

Figure 3.4: Example of recommendations from different methods. See the main text in Section 3.6.4 for description.

The trajectory recommended by PersTour is a tour with 11 POIs, as shown in Figure 3.4a, with none of the desired intermediate POIs visited. PoiRank (Figure 3.4b) recommended a tour with correct POIs, but with completely different routes. On the other hand, Markov (Figure 3.4c) missed one POI but one of the intermediate routes is consistent with the ground truth. The best recommendation, as shown in Figure 3.4d, with exactly the same points and routes as the ground truth, which in this case is achieved by Rank+MarkovPath.

3.7 Summary

We propose approaches that jointly optimising point preferences and routes to recommend trajectories. This is in contrast to related work which looks at only POI or next location recommendation. Point preferences are learned by ranking according to POI and query features, and factorised transition probabilities between POIs are learned from previous trajectories extracted from social media. We investigate the maximum likelihood sequence approach (which may recommend sub-tours) and propose an improved method. Our feature driven approach naturally allows learning the combination of POI ranks and routes. We argue that one should measure performance with respect to the visiting order of POIs, and suggest a new pairs-F_1 metric. We empirically evaluate our tour recommendation approaches on five datasets extracted from Flickr photos, and demonstrate that our methods improves on prior work, in terms of both the traditional F_1 metric and our proposed performance measure. Our promising results from learning points and routes for trajectory recommendation suggests that research in this domain should consider both information sources simultaneously.

Chapter 4

Structured Recommendation for Travel Trajectories

We investigated the path recommendation problem for travel trajectories in Chapter 3. Here, we continue the study of this problem. In contrast to the feature-driven approach that independently learns POI preferences and transition patterns (Section 3.5.2), we propose methods that systematically incorporate both of these information sources by leveraging a substantially modified SSVMs (Section 2.3). Section 4.2 casts path recommendation as a structured prediction problem, and analyses the fundamental challenges in recommending trajectories. Section 4.3 presents a structured recommendation approach, which is based on the SSVMs, but with a loss function that accounts for the existence of multiple ground truths, as well as updated training and prediction procedures for recommending paths. Empirical results demonstrate the effectiveness of the proposed approach for travel trajectory recommendation (Section 4.4).

4.1 Introduction

Established techniques for recommendation have focused on a fixed set of individual items such as books or movies (Linden et al., 2003; Koren, 2010; Agarwal et al., 2013; Amatriain and Basilico, 2015; Gomez-Uribe and Hunt, 2015). This does not however capture scenarios where the content is naturally organised with *structures* (e.g., sequence, graph or set). For example, consider recommending linked websites for e-commerce (Antikacioglu et al., 2015), a chemical compound (Dehaspe et al., 1998; Agrafiotis et al., 2007), or a playlist of songs (McFee and Lanckriet, 2011; Chen et al., 2012; Hidasi et al., 2015; Choi et al., 2016). Recall that the problem of recommending travel trajectories is to suggest a trajectory of points-of-interest (POIs) in a city to a visitor (Lu et al., 2010, 2012; Lim et al., 2015; Chen et al., 2016; He et al., 2018), i.e., a sequence of POIs without repeats, which is also known as a *path*.

Table 4.1: Challenges of travel trajectory recommendation and the proposed solutions.

Challenge	Solution
C1. Global cohesion	Structured support vector machines (§4.3.2)
C2. Multiple ground truths	Ground truths aggregation in structured hinge loss (§4.3.3)
C3. Loop elimination	The list Viterbi algorithm or integer linear programming (§4.3.4)

4.2 Problem statement

Suppose we have a set of points-of-interest (POIs) $\mathcal{P}$ in a city and historical trajectories visited by travellers $\mathcal{S} = \{(\mathbf{x}^{(i)}, \{\mathbf{y}^{(ij)}\}_{j=1}^{N_i})\}_{i=1}^{N}$, where each $\mathbf{x}^{(i)}$ is a distinct query with $\{\mathbf{y}^{(ij)}\}_{j=1}^{N_i}$ an associated set of observed trajectories. Here we define a *trajectory query* as $\mathbf{x} = (p_s, L)$ that comprises the start point $p_s \in \mathcal{P}$ as well as the trip length L (i.e., the desired number of POIs, including p_s). Similar to Chapter 3, our task is to learn a recommender from $\mathcal{S}$ that can suggest trajectories for a new query not seen in $\mathcal{S}$. Compared to the definition of trajectory query in Section 3.2, the end location is omitted here, with the aim of accommodating broader scenarios of travel trajectory recommendation in practice. As a remark, we expect most queries to have several distinct trajectories; minimally, for example, there may be two nearby POIs that are visited in interchangeable order by different travellers.

Travel trajectory recommendation brings several challenges, the most immediate of which is the need to ensure *global cohesion* of recommendations. To illustrate, consider a naïve approach which ignores all sequential structure: we could learn a user's preference for individual POIs, and create a trajectory based on the top ranked items. Such an approach may be sub-optimal, as it is unlikely e.g., a user will want to visit three restaurants in a row; more generally, while a user's two favourite songs might be in the metal and country genres, a playlist featuring these songs in succession may be jarring. To effectively ensure such global cohesion, we propose to attack the travel trajectory recommendation problem via structured prediction, in particular, leveraging the toolkit of the SSVMs (Section 2.3.1).

However, a vanilla application of such methods does not suffice, owing to two additional challenges: in trajectory recommendation, each input can have *multiple ground truths*, since multiple trajectories may be reasonable for a single query; and further, one needs to constrain predictions to avoid *repeated* elements, since users are unlikely to wish to visit the same POI twice. We nonetheless show how to extend the SSVMs to address these challenges, via a novel application of the list Viterbi algorithm (Section 2.4.2), or alternatively, by adapting an integer linear programming formulation of the s-t path TSP (Section 2.4.3). Table 4.1 summaries the three challenges of travel trajectory recommendation and the solutions proposed in this chapter.

4.3 A structured recommendation approach

In this section, we first cast travel trajectory recommendation as a structured prediction problem, then develop methods to address the three challenges for this task, followed by an extensive discussion of different design variants and practical choices.

4.3.1 Trajectory recommendation as structured prediction

Recall that structured prediction is to predict a structured label $\mathbf{y} \in \mathcal{Y}$ according to a learned score function $f(\mathbf{x}, \mathbf{y})$ for an input $\mathbf{x} \in \mathcal{X}$ (Section 2.3). We observe that the task of recommending a travel trajectory can be cast as a structured prediction problem: given query $\mathbf{x}$, and a suitable scoring function f, we wish to find

$$\mathbf{y}^* = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}} \; f(\mathbf{x}, \mathbf{y}), \tag{4.1}$$

where $\mathcal{Y}$ is the space of all possible trajectories with POIs in $\mathcal{P}$ that conform to the constraints imposed by the query $\mathbf{x}$. In particular, $\mathbf{y} = (y_1, y_2, \ldots, y_L)$ is a trajectory with L POIs and $y_1 = p_s$.

We remark that one cannot naïvely compute $\operatorname{argmax}_{\mathbf{y}} f(\mathbf{x}, \mathbf{y})$ as in typical recommender systems, since it is often intractable to enumerate all possible trajectories in a city. Further, the inability to efficiently enumerate $\mathbf{y}$ in (4.1) also poses a challenge in designing a suitable score function $f(\mathbf{x}, \mathbf{y})$, e.g., matrix factorisation would require associating a latent feature with each $\mathbf{y}$, which will be infeasible.

We suppose the scoring function $f(\mathbf{x}, \mathbf{y})$ takes the form of a linear function, i.e., $f(\mathbf{x}, \mathbf{y}) = \mathbf{w}^\top \Psi(\mathbf{x}, \mathbf{y})$, where $\mathbf{w}$ is a weight vector, and $\Psi(\mathbf{x}, \mathbf{y})$ is a joint feature map of query $\mathbf{x}$ and trajectory $\mathbf{y}$. To specify the joint feature map, we consider the unary terms for each POI in $\mathbf{y}$, i.e., $y_1, \ldots, y_L$, as well as pairwise interactions between adjacent POIs in $\mathbf{y}$, i.e., y_l and y_{l+1} for $l \in \{1, \ldots, L-1\}$. Subsequently, $f(\mathbf{x}, \mathbf{y})$ decomposes into a weighted sum of each of these features:[1]

$$f(\mathbf{x}, \mathbf{y}) = \mathbf{w}^\top \Psi(\mathbf{x}, \mathbf{y}) = \sum_{l=1}^{L} \mathbf{w}_l^\top \psi_l(\mathbf{x}, y_l) + \sum_{l=1}^{L-1} \mathbf{w}_{l,l+1}^\top \psi_{l,l+1}(\mathbf{x}, y_l, y_{l+1}), \tag{4.2}$$

where ψ_l is a feature map between the input query $\mathbf{x}$ and one output label element y_l, with a corresponding weight vector $\mathbf{w}_l$; and $\psi_{l,l+1}$ is a pairwise feature vector that captures the interactions between consecutive labels y_l and y_{l+1}, with a corresponding weight vector $\mathbf{w}_{l,l+1}$. In practice, to deal with data sparsity, we could share the weights among POIs as well as the weights among POI-POI interactions, i.e., $\mathbf{w}_l = \mathbf{w}_{\text{unary}}$ where $l \in \{1, \ldots, L\}$ and $\mathbf{w}_{l,l+1} = \mathbf{w}_{\text{pairwise}}$ for $l \in \{1, \ldots, L-1\}$.

[1] Appendix D.1 details the feature maps.

4.3.2 Global cohesion and the SP model

Trajectory recommendation can be plausibly solved with approaches that do not exploit the structure inherent in the outputs. While such approaches can certainly be useful, their modelling power is inherently limited, as they cannot ensure the *global cohesion* of the corresponding recommendations. For example, a ranking based approach might find three restaurants to be the highest scoring POIs; however, it is unlikely that most travellers will enjoy this.

We propose to address challenge **C1** by learning an SSVMs from the historical trajectories. To temporarily mitigate the challenge of multiple ground truths, i.e., there is a set of trajectories $\{\mathbf{y}^{(ij)}\}_{j=1}^{N_i}$ for query $\mathbf{x}^{(i)}$, we adopt a straightforward approach that creates N_i examples $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(ij)})\}_{j=1}^{N_i}$, and then feed this to the standard SSVMs:

$$\begin{aligned} &\min_{\mathbf{w},\xi\geq 0} \frac{1}{2}\mathbf{w}^\top\mathbf{w} + \frac{C}{N'}\sum_{i=1}^{N'}\xi^{(i)} \\ &s.t.\ \ f(\mathbf{x}^{(i)},\mathbf{y}^{(i)}) - f(\mathbf{x}^{(i)},\bar{\mathbf{y}}) \geq \Delta(\mathbf{y}^{(i)},\bar{\mathbf{y}}) - \xi^{(i)},\ i\in\{1,\ldots,N'\},\ \forall\bar{\mathbf{y}}\in\mathcal{Y}. \end{aligned} \tag{4.3}$$

Here $N' = \sum_{i'=1}^{N} N_{i'}$ and $\Delta(\mathbf{y},\bar{\mathbf{y}})$ measures the discrepancy between $\bar{\mathbf{y}}$ and $\mathbf{y}$; slack variable $\xi^{(i)}$ is the structured hinge loss of the i-th example (Equation 2.7). Alternatively, we can use *one* slack variable to represent the sum of the N' hinge losses in (4.3), as described in Section 2.3.1. We call this method the *structured prediction* (SP) model.

Note that the loss-augmented inference of the SP model is

$$\operatorname*{argmax}_{\bar{\mathbf{y}}\in\mathcal{Y}} \left\{\Delta(\mathbf{y}^{(i)},\bar{\mathbf{y}}) + f(\mathbf{x}^{(i)},\bar{\mathbf{y}})\right\}, \tag{4.4}$$

when the underlying graph of the SSVMs is a sequence or tree, the loss-augmented inference may be solved efficiently if Δ is decomposable with respect to individual and pairs of label elements, e.g., using the Viterbi algorithm (Joachims et al., 2009b).

The naïve approach of generating a labelled example for each observed trajectory of a query will result in contradictory constraints during training, i.e., each of the observed trajectories of the same query is required to have a larger score than others. This limitation will be addressed in the next section.

4.3.3 Multiple ground truths and the SR model

The SP model is appealing due to its simplicity. However, it is sub-optimal: the result of loss-augmented inference on $(\mathbf{x}^{(i)}, \mathbf{y}^{(ij)})$ could be a ground truth label $\mathbf{y}' \in \{\mathbf{y}^{(ij)}\}_{j=1}^{N_i}$, which means we would incorrectly penalise predicting $\mathbf{y}'$ for query $\mathbf{x}^{(i)}$. To overcome this limitation of the SP model, we modify the structured hinge loss of the SSVMs to

explicitly exclude the set of ground truths for the i-th example,

$$\ell^{\text{SR}}(\mathbf{y}^{(i)}, f(\mathbf{x}^{(i)}, \cdot)) = \max\left(0,\ \Delta(\mathbf{y}^{(i)}, \bar{\mathbf{y}}^{(i)}) - \left(f(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) - f(\mathbf{x}^{(i)}, \bar{\mathbf{y}}^{(i)})\right)\right), \quad \bar{\mathbf{y}}^{(i)} \in \mathcal{Y} \setminus \{\mathbf{y}^{(ij)}\}_{j=1}^{N_i}. \tag{4.5}$$

The modified structured hinge loss (4.5) results in the following "n-slack" SSVMs:

$$\begin{aligned} \min_{\mathbf{w}, \boldsymbol{\xi} \geq 0}\ & \frac{1}{2}\mathbf{w}^\top \mathbf{w} + \frac{C}{N'} \sum_{i=1}^{N} \sum_{j=1}^{N_i} \xi^{(ij)} \\ s.t.\ \ & f(\mathbf{x}^{(i)}, \mathbf{y}^{(ij)}) - f(\mathbf{x}^{(i)}, \bar{\mathbf{y}}^{(i)}) \geq \Delta(\mathbf{y}^{(ij)}, \bar{\mathbf{y}}^{(i)}) - \xi^{(ij)}, \\ & \bar{\mathbf{y}}^{(i)} \in \mathcal{Y} \setminus \{\mathbf{y}^{(ij)}\}_{j=1}^{N_i},\ i \in \{1, \ldots, N\},\ j \in \{1, \ldots, N_i\}. \end{aligned} \tag{4.6}$$

where $N' = \sum_{i=1}^{N} N_i$, and the "1-slack" SSVMs of (4.6) can be formed accordingly.

Intuitively, the loss function of this approach (Equation 4.5) resembles a ranking objective, as the constraint for $\bar{\mathbf{y}}^{(i)}$ enforces that the positively labelled items (those that the user likes) are scored higher than all other unseen items. Such objectives have been proven useful in classic unstructured recommendation (Rendle et al., 2009). Compared to the SP model (4.3), the key distinction is that (4.6) explicitly *aggregates* all the ground truth labels for each input when generating constraints, and we call this method that addresses challenge **C2** the *structured recommendation* (SR) model.

As a remark, the loss-augmented inference of the SR model is

$$\operatorname*{argmax}_{\bar{\mathbf{y}}^{(i)} \in\ \mathcal{Y} \setminus \{\mathbf{y}^{(ij)}\}_{j=1}^{N_i}} \left\{\Delta(\mathbf{y}^{(ij)}, \bar{\mathbf{y}}^{(i)}) + f(\mathbf{x}^{(i)}, \bar{\mathbf{y}}^{(i)})\right\}. \tag{4.7}$$

In this way, we do not have contradictory constraints where two ground truth labels are each required to have larger score than the other.

4.3.4 Eliminating loops in recommendation

This section addresses challenge **C3**, i.e., eliminate loops for trajectory recommendation, since it is desirable that the recommended sequence consists of distinct POIs, or be a *path* in the candidate space (e.g., locations). In particular, for a learned SP or SR model, we make a recommendation for query $\mathbf{x}$ by computing

$$\mathbf{y}^* = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{Y}_{\text{path}}} f(\mathbf{x}, \mathbf{y}), \tag{4.8}$$

where $\mathcal{Y}_{\text{path}} \subseteq \mathcal{Y}$ comprises all elements of $\mathcal{Y}$ that are paths. However, classic structured prediction does not constrain the output sequence, and having such a

path constraint makes both inference and learning harder. Suppose the underlying sequence model of the learned SP and SR model satisfy the Markov assumption (Section 2.4.1), we can leverage the list Viterbi algorithm or the integer linear programming formulation of the s-t path TSP to recommend paths.

Given a score function that can be decomposed into unary and pairwise scores such as (4.2), the list Viterbi algorithm (LVA) finds the K highest scoring sequences (Section 2.4.2). Here we employ the variant of LVA that can sequentially find the k-th best scoring sequence given the best, second best, ..., $(k-1)$-th best scoring sequences (Seshadri and Sundberg, 1994; Nilsson and Goldberger, 2001), which ideally resolves the scenario when the number of sequences to find is unknown a priori. This is the case when recommending a path given a trained SP or SR model, since we have to keep searching the next-best scored sequence until a path is discovered.

Alternatively, we can adapt the ILP formulation of the s-t path TSP (Section 2.4.3) and leverage highly optimised off-the-shelf ILP solvers to find a highest scored path. In particular, we first modify the constraints (2.35)-(2.38) by introducing additional binary variables v_i, $i \in \{1, \ldots, M\}$ where $M = |\mathcal{P}|$ is the number of POIs in a city, to indicate if a POI is the end point of a path, then optimise an objective that incorporates both the unary and pairwise scores:

$$
\begin{aligned}
\max_{\mathbf{z},\mathbf{u},\mathbf{v}} \; & \sum_{k=1}^{M} \mathbf{w}_k^\top \psi_k(\mathbf{x}, p_k) \sum_{j=1}^{M} z_{jk} + \sum_{j,k=1}^{M} z_{jk} \mathbf{w}_{jk}^\top \psi_{jk}(\mathbf{x}, p_j, p_k), \\
s.t. \; & \sum_{k=2}^{M} z_{1k} = 1, \quad \sum_{j=2}^{M} z_{j1} = v_1 = 0, \\
& \sum_{j=1}^{M} \sum_{k=1}^{M} z_{jk} = L - 1, \quad \sum_{j=1}^{M} z_{jj} = 0, \\
& \sum_{j=1}^{M} z_{ji} = v_i + \sum_{k=2}^{M} z_{ik} \leq 1, \quad i \in \{2, \cdots, M\} \\
& u_j - u_k + 1 \leq (M-1)(1 - z_{jk}), \quad j, k \in \{2, \cdots, M\}.
\end{aligned}
\tag{4.9}
$$

Note the binary variable z_{jk} indicates whether p_k will be immediately visited following the visit of p_j, and integer variables $\mathbf{u}$ track the ranks of POIs in a trajectory, which are employed to eliminate sub-tours (Miller et al., 1960).

4.3.5 SP and SR model training

We have described how one can recommend a trajectory (which is a *path*) given a trained SP or SR model, a natural follow up question is how one can efficiently train them. A major challenge in learning the SP and SR model is performing the loss-

augmented inference. Note that the SP model can be trained as per the vanilla SSVMs, where the loss-augmented inference[2] (Equation 4.4) may be solved with the classic Viterbi algorithm. However, this approach is not applicable to the loss-augmented inference of the SR model (Equation 4.7), since the best-scoring sequence from the above approach could be in the ground-truth set.

One approach to solve the loss-augmented inference of the SR model involves the use of the LVA. In particular, for query $\mathbf{x}^{(i)}$ and an associated trajectory $\mathbf{y}^{(ij)}$, we solve $\text{argmax}_{\bar{\mathbf{y}}^{(i)} \in \mathcal{Y}} \left\{ \Delta(\mathbf{y}^{(ij)}, \bar{\mathbf{y}}^{(i)}) + f(\mathbf{x}^{(i)}, \bar{\mathbf{y}}^{(i)}) \right\}$ using the LVA by keeping decoding the next best sequence until we find one that is not in $\{\mathbf{y}^{(ij)}\}_{j=1}^{N_i}$. Note that the LVA can be used for loss-augmented inference with Δ be the Hamming loss, the most common loss function for sequence prediction tasks, since LVA only requires Δ be decomposable with respect to the individual elements of a structured label.

As a remark, we have not required the result of loss-augmented inference to be a path for both the SP and the SR model, we discuss how one can enforce the *path* constraint for training in the next section.

4.3.6 Discussion

We have shown how one can train the SP and SR models, as well as make recommendations using a learned model. In this section, we discuss many alternative approaches that might be plausible for the SP and SR models training and recommendation.

Eliminating loops for training: SPpath and SRpath

The list Viterbi algorithm can also enforce that loss-augmented inference only considers sequences that are *paths*, e.g., by checking if each of the next-best sequences has a loop. This idea can be applied to both the SP and SR models, as enforcing path constraints is independent of excluding multiple ground truths. We call the resulting models SPpath and SRpath respectively.

Eliminating multiple ground truths with ILP?

A natural question is whether one can use the ILP approach to exclude observed trajectories when training an SR model, i.e., solving the loss-augmented inference (4.7). In fact, this can be done as long as the loss $\Delta(\mathbf{y}, \bar{\mathbf{y}})$ can be represented as a linear function of variables $\mathbf{z}$ and $\mathbf{u}$ in (4.9). One example is the number of mis-predicted POIs disregarding the order $\Delta(\mathbf{y}, \bar{\mathbf{y}}) = \sum_{l=2}^{L}(1 - \sum_{k=1}^{M} z_{k,y_l})$. However, we note that

[2]We again assume the underlying sequence model satisfies the Markov assumption.

Hamming loss is not applicable here, as $\Delta(\mathbf{y}, \bar{\mathbf{y}}) = \sum_{l=1}^{L}(1 - [\![y_l = \bar{y}_l]\!])$ cannot be expressed as a linear function of variables $\mathbf{z}$ and $\mathbf{u}$.

Recommending more than one trajectory

Since multiple possible trajectories can start at the same POI, it is desirable to predict multiple trajectories for a query. The LVA is naturally fitted for this problem. It turns out that the ILP approach can also be applied to this task, the idea is to incorporate those decoded paths into additional constraints and then find the top-K scored paths in a sequential manner. In particular, given the $K-1$ top scored paths $\{\mathbf{y}^{(k)}\}_{k=1}^{K-1}$, the K-th best scored path can be found by solving the ILP (4.9) but with the following additional constraints:

$$\sum_{l=1}^{L-1} z_{y_l, y_{l+1}} \leq L - 2, \quad \forall \mathbf{y} \in \{\mathbf{y}^{(k)}\}_{k=1}^{K-1}. \tag{4.10}$$

Practical choices: ILP vs LVA vs other heuristics

When performing recommendations for SP and SR model, we found that state-of-the-art ILP solvers converge to a solution faster than the LVA if the trajectory length L is large. The reason is, while the LVA is a polynomial time algorithm given the list depth K (Nilsson and Goldberger, 2001), in reality K is unknown a priori and can be very large for long trajectories. We therefore use ILP for very long ($L \geq 10$) trajectories in the experiments (otherwise the LVA is employed).

One might also consider the well-known Christofides algorithm (Christofides, 1976) for recommending paths, as this is known to generate a solution within a factor of 3/2 of the optimal solution for the TSP. However, the resulting path will have less than the desired number of POIs, and its score will not be optimal.

Incorporating time constraints

We can adapt the ILP formulation (4.9) to incorporate time constraints (e.g., time budget for a desired trajectory, time-of-the-day constraints on POIs) by employing an approach similar to the one presented in Section 3.5.4. In particular, we can specify a time budget T in a trajectory query instead of the number of desired POIs L. As a result, we can replace the trip length constraint in (4.9) with the following inequality that constrains the time budget of a desired trajectory

$$\sum_{j=1}^{M}\sum_{k=2}^{M} z_{jk}\left(t_{jk} + \frac{1}{2}(c_j + c_k)\right) + \frac{1}{2}\left(c_1 + \sum_{j=2}^{M} v_j c_j\right) \leq T, \tag{4.11}$$

Table 4.2: Summary of challenges considered in different methods.

	Global Cohesion	Multi-GTs	Loop Elim. (Train)	Loop Elim. (Test)
SP	√	×	×	√
SPPATH	√	×	√	√
SR	√	√	×	√
SRPATH	√	√	√	√

where c_j, $j \in \{1, \ldots, M\}$ denotes the time to be spent at POI p_j and t_{jk}, $j, k \in \{1, \ldots, M\}$ is the travelling time from p_j to p_k, which can either be specified by users or can be computed from existing travel data.

In addition, this method can be further extended to incorporate the time-of-the-day constraints on POIs (i.e., a POI can only be visited in a limited time frame in a day), e.g., by employing an approach similar to the one presented in Appendix C.

4.3.7 Summary of proposed methods

Table 4.2 summarises the methods proposed in this chapter for path recommendation. The SP method employ the standard SSVMs to recommend paths, without accounting for multiple ground truths for each input query; the SR method, on the other hand, incorporates this fact. Both SP and SR eliminate loops when making recommendations, and their PATH variants, SPPATH and SRPATH, further eliminate loops in training.

4.4 Experiments

We present empirical evaluations for the trajectory recommendation task on real-world datasets of photo tours, created from the publicly available YFCC100M corpus (Thomee et al., 2016) as described below.

4.4.1 Photo trajectory datasets

We used the trajectory data extracted from Flickr photos for the cities of Osaka, Glasgow, Toronto, Edinburgh and Melbourne (Lim et al., 2015; Chen et al., 2016; De Choudhury et al., 2010). Each dataset comprises of a list of trajectories, being a sequence of points-of-interest (POIs), as visited by various Flickr users and recorded by the geotags in photos. Table 4.3 summarises the profile of each dataset. We see that most queries have more than one ground truth, making our recommendation setting relevant. Further, each query has an average of 4-9, and a maximum of 30-100 trajectories, as shown in Figure 4.1. The histograms of trajectory length are shown

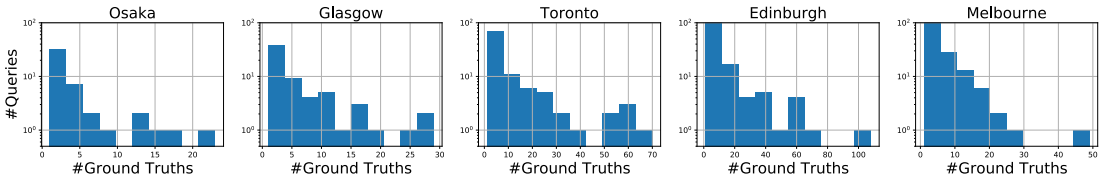


Figure 4.1: Histograms of the number of trajectories per query.

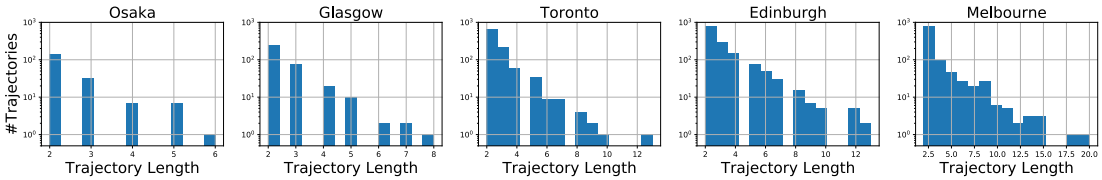


Figure 4.2: Histograms of trajectory length.

Table 4.3: Statistics of trajectory datasets. Including the number of trajectories, POIs, users and queries; the number of queries with a single, 2 to 5, and more than 5 ground truths; profile of trajectory length, i.e., less than 5 (short trajectories) and more than 5 POIs (long trajectories).

	Osaka	Glasgow	Toronto	Edinburgh	Melbourne
Trajectories	186	351	977	1,412	1,018
POIs	26	25	27	28	84
Users	130	219	454	677	456
Queries	47	64	99	147	280
Ground Truths (1)	17	23	30	43	141
Ground Truths (2 to 5)	22	22	33	50	88
Ground Truths (> 5)	8	19	36	54	51
Short Trajectories	178	336	918	1,224	879
Long Trajectories	8	15	59	188	139

in Figure 4.2. In all datasets, each user has on average about two trajectories. This makes user-specific recommendation impractical, and also undesirable because a user would want different recommendations given different starting locations, and not a static recommendation no matter where she is. The sparsity of this dataset presents a barrier for large-scale evaluations.

4.4.2 Evaluation setting

We compare the propose methods to the following four baselines:

- The RANDOM baseline recommends a sequence of POIs by sampling uniformly at random from the whole set of POIs (without utilising any POI or query related features).

- The stronger POPULARITY baseline recommends the top-L most popular POIs, i.e., the POIs visited by the most number of users in the training set, which is independent of query.
- POIRANK (Section 3.5.1) augments POPULARITY with a number of POI-query features (see Appendix D.1.1), and trains a RankSVM to learn a score for each POI for a given query. A trajectory is constructed from the top-L scored POIs.
- MARKOV (Section 3.5.1) learns a Markov chain by factorising the transition probabilities between POIs according to pairwise features (see Appendix D.1.2).

We consider four variants of path recommendation, starting with a structured prediction model, then incorporating multiple ground truths, and finally enforcing path constraints for training:

- The SP and SR methods, using both POI-query features and pairwise features (see Appendix D.1).
- SPPATH and SRPATH, using the same features as SP and SR, but with path constraints for model learning.

We reiterate that irrespective of the training procedure, the SP, SR, SPPATH and SRPATH all recommend paths. Among the baseline approaches, the RANDOM, POPULARITY, POIRANK recommend paths; the MARKOV, which leverages the vanilla Viterbi algorithm, may recommend sequences with repeated POI visits (i.e., *walks*).

We evaluate each algorithm using leave-one-query-out cross validation. That is, holding out all the relevant trajectories for each query $\mathbf{x}^{(i)}$ (i.e., $\{\mathbf{y}^{(ij)}\}_{j=1}^{N_i}$) in each round. The regularisation constant C is tuned using Monte Carlo cross validation on the training set. We use three performance measures for POIs, sequences and ordered lists. The **F_1 score on points** (Lim et al., 2015) computes the F_1 score on the predicted versus seen points without considering their relative order. The **F_1 score on pairs** (Chen et al., 2016) mitigate this by computing the F_1 score on all ordered pairs in the predicted versus ground truths. The well-known rank correlation **Kendall's τ** (Agresti, 2010) computes the ratio of concordant (correctly ranked) minus discordant pairs, over all possible pairs after accounting for ties. See Appendix D.2 for details.

Path recommendation methods perform ranking on a very large label set (of size $|\mathcal{P}|^L$). We report results on the *best of top k* (Russakovsky et al., 2015): for all methods, we predict the top k trajectories and then report the best match of any in the top k to any trajectory in the ground truth set. To obtain top-k recommendations with RANDOM, we independently repeat it k times. To perform top-k recommendation with POPULARITY, POIRANK and MARKOV, we employ the list Viterbi algorithm. For

POPULARITY, the score of a path is the accumulated popularity of all POIs in the path; for POIRANK and MARKOV, the score of a path is the likelihood (the ranking scores for POIs are first transformed into a probability distribution using the softmax function, as described in Section 3.5.2).

4.4.3 Results and discussion

Table 4.4, 4.5 and 4.6 summarises the performance of all methods for top-10 recommendations in terms of the three evaluation metrics respectively. We remark the performance values consist of the *mean* and *standard error* of the corresponding metric. We also compare the performance for all values of top-k with $k = 1, \ldots, 10$, a selection of the curves for `Glasgow` is shown in Figure 4.3, 4.4 and 4.5. Additional experimental results across all five datasets are available in Appendix D.3.

Is exploiting structure helpful? We can observe from the results in Table 4.4 – 4.6 that POIRANK, MARKOV and SP methods that convert the trajectory recommendation task into data that is amenable to off-the-shelf methods (ranking, Markov chains and structured prediction respectively) performs better than RANDOM and POPULARITY baselines but are not the best performing methods. Although MARKOV achieves the best performance on the smallest `Osaka` dataset, the best results on other (larger) datasets are obtained using our proposed methods SR and SRPATH, especially for long trajectories (Figure 4.3 – 4.5).

Is imposing the path constraint helpful? Recall that the SP, SR methods and their PATH variants impose the path constraint for recommendation. We observe that these methods perform better than other baselines (except on the smallest `Osaka` dataset), which suggest that it is beneficial to impose path constraint for recommendation.

However, the situation is rather different for training. SPPATH and SRPATH take significantly more time (to train) than their non-PATH variants, since the repeated invoked loss-augmented inference procedures (Eq. 4.4 and Eq. 4.7) have to search for paths. Further, the cost of training SPPATH and SRPATH is exacerbated when trajectories (in the training set) become longer. This is the main reason that we do not have results for SPPATH and SRPATH on the `Edinburgh` and `Melbourne` datasets, since we cannot train these two methods in reasonable time due to a significantly larger portion of long trajectories in these two datasets, as shown in Table 4.3.

On the other hand, the overall performances of SPPATH and SRPATH are comparable to those of their non-PATH variants (Table 4.4 – 4.6), and this is also the case for recommending short trajectories (Figure 4.3a – 4.5a).

Table 4.4: F_1 score on points of trajectory recommendation methods on best of top-10. The **best** method for each dataset is shown in bold, the *second best* is shown in italic.

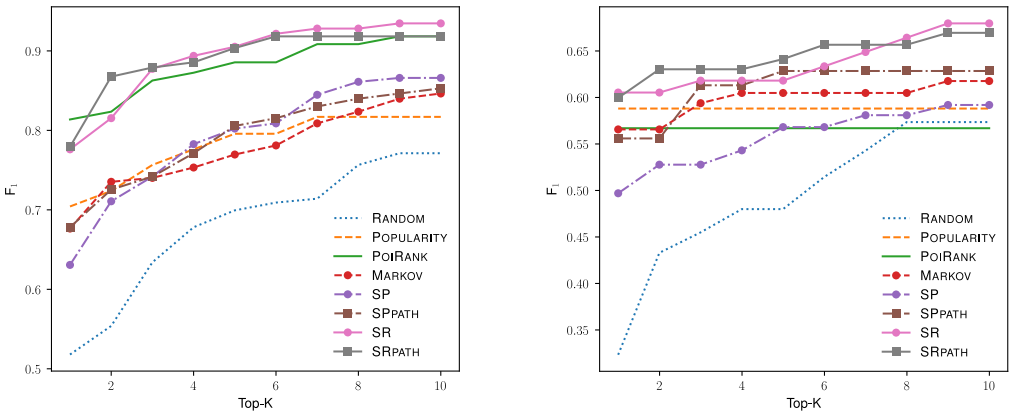
	Osaka	Glasgow	Toronto	Edinburgh	Melbourne
Random	.703 ± .032	.731 ± .026	.696 ± .021	.689 ± .014	.511 ± .012
Popularity	.786 ± .034	.771 ± .033	.746 ± .022	.768 ± .017	.587 ± .016
PoiRank	.804 ± .034	.847 ± .025	.807 ± .020	.754 ± .017	.609 ± .017
Markov	**.840** ± .029	.800 ± .028	.819 ± .019	.679 ± .021	.508 ± .017
SP	.770 ± .039	.810 ± .027	.733 ± .023	*.796* ± .016	*.622* ± .016
SPpath	.809 ± .033	.807 ± .026	.755 ± .022	N/A	N/A
SR	.793 ± .033	**.883** ± .023	**.828** ± .019	**.809** ± .015	**.628** ± .016
SRpath	*.820* ± .031	*.868* ± .023	*.823* ± .020	N/A	N/A

Table 4.5: F_1 score on pairs of trajectory recommendation methods on best of top-10. The **best** method for each dataset is shown in bold, the *second best* is shown in italic.

	Osaka	Glasgow	Toronto	Edinburgh	Melbourne
Random	.451 ± .057	.495 ± .046	.438 ± .034	.411 ± .024	.205 ± .018
Popularity	.626 ± .055	.623 ± .051	.546 ± .036	.561 ± .027	.341 ± .023
PoiRank	.661 ± .056	.726 ± .043	.646 ± .034	.557 ± .027	.388 ± .023
Markov	**.693** ± .051	.635 ± .048	.644 ± .033	.472 ± .030	.268 ± .023
SP	.620 ± .061	.658 ± .046	.530 ± .037	*.588* ± .027	*.390* ± .023
SPpath	.664 ± .055	.648 ± .045	.552 ± .036	N/A	N/A
SR	.637 ± .055	**.770** ± .039	**.660** ± .033	**.608** ± .026	**.403** ± .023
SRpath	*.671* ± .053	*.746* ± .041	*.656* ± .034	N/A	N/A

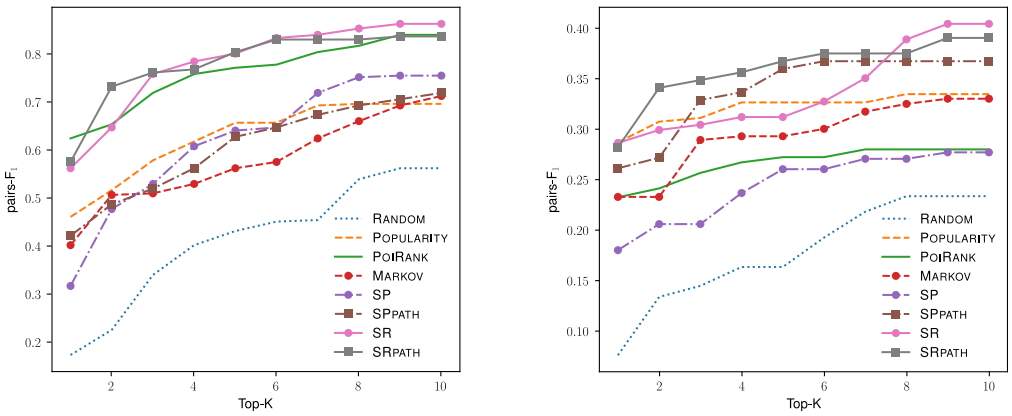
Table 4.6: Kendall's τ of trajectory recommendation methods on best of top-10. The **best** method for each dataset is shown in bold, the *second best* is shown in italic.

	Osaka	Glasgow	Toronto	Edinburgh	Melbourne
Random	.685 ± .035	.703 ± .029	.652 ± .024	.616 ± .018	.488 ± .013
Popularity	.768 ± .038	.748 ± .036	.719 ± .024	.718 ± .020	.567 ± .017
PoiRank	.787 ± .037	.830 ± .029	.784 ± .023	.712 ± .020	.601 ± .017
Markov	**.824** ± .031	.781 ± .031	.789 ± .022	.739 ± .018	.541 ± .016
SP	.749 ± .043	.790 ± .030	.697 ± .027	*.741* ± .020	*.602* ± .017
SPpath	.791 ± .036	.787 ± .029	.719 ± .026	N/A	N/A
SR	.777 ± .036	**.868** ± .026	**.802** ± .022	**.761** ± .018	**.610** ± .017
SRpath	*.803* ± .034	*.853* ± .026	*.797* ± .022	N/A	N/A



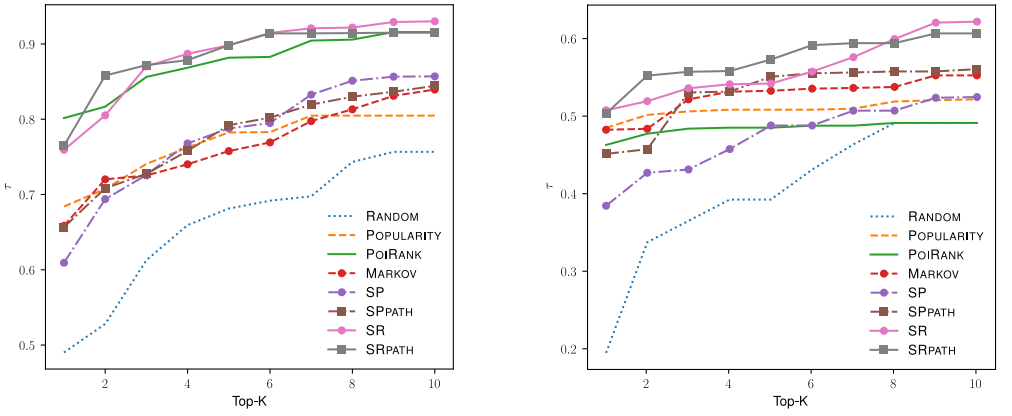
(a) Short Trajectories (b) Long Trajectories

Figure 4.3: F_1 score on points over $k = 1:10$ for short and long trajectories on Glasgow.



(a) Short Trajectories (b) Long Trajectories

Figure 4.4: F_1 score on pairs over $k = 1:10$ for short and long trajectories on Glasgow.



(a) Short Trajectories (b) Long Trajectories

Figure 4.5: Kendall's τ over $k = 1:10$ for short and long trajectories on Glasgow.

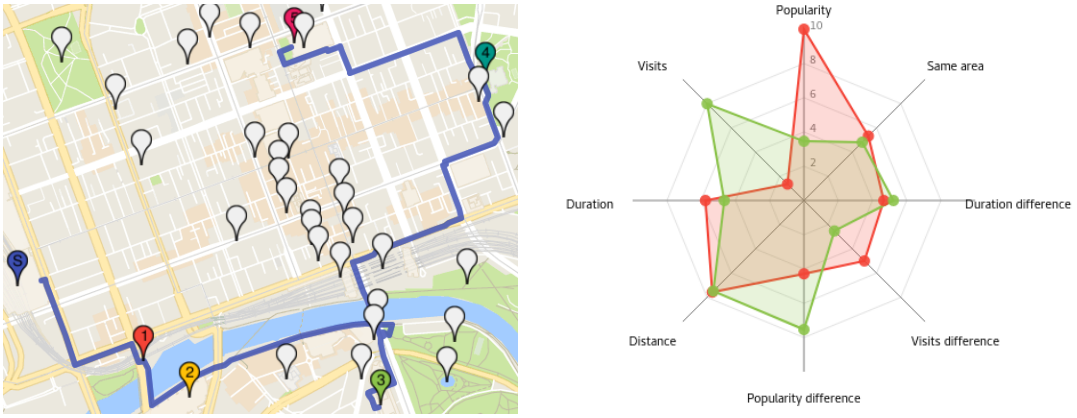


Figure 4.6: Example of a recommended trajectory with POI sequence $(S, 1, 2, 3, 4, 5)$ (left); Radar charts that compare scores of two POIs (red chart for POI 1, green chart for POI 3) decomposed according to eight features (right): *Popularity* and *Distance* contribute the most to the unary score of POI 1; *Distance*, *Visits* and *Popularity Difference* dominate the contribution to the unary score of POI 3.

When recommending long trajectories, SPPATH demonstrates advantages over SP, however, the significantly better performance achieved by SR, which is comparable to that of SRPATH, as shown in Figure 4.3b – 4.5b, makes SR the best practical choice for path recommendation.

Is it important to account for multiple ground truths? We observe that SR always performs better than SP, and similarly for the PATH variants of both methods. This indicates that our first extension – explicitly modelling multiple ground truths – is important to achieve good performance. We can also see that the advantages of SR and SRPATH are salient for longer trajectories (Figure 4.3b – 4.5b), where pairwise and sequential information play a larger role.

Overall, path recommendation methods have shown superior performance in travel trajectory recommendation tasks on a public benchmark. Most notably, taking into account multiple ground truths in structured prediction and modelling path constraints are important; and the effects are more pronounced in longer trajectories.

4.4.4 A qualitative example

Figure 4.6 shows a trajectory recommended by the SR model in `Melbourne`, the trajectory $(S, 1, 2, 3, 4, 5)$ includes six POIs and is visualised on a map (left); the unary scores of two POIs (Eq. 4.2) are decomposed according to eight POI features (right), in particular, the red radar chart shows the decomposition of the unary score of POI 1 (red marker on map), and the green chart shows the decomposition of the unary score of POI 3 (green marker on map). We can see that popularity and distance

related features (i.e., *Popularity* and *Visits difference*, *Distance* and *Same area*) are the main contributors to POI 1; and POI 3 is largely supported by distance and visits related features (i.e., *Distance*, *Visits* and *Popularity difference*).

4.5 Related work

We summarise recent work most related to the recommendation of structured objects, in particular the problem of recommending travel trajectories, predicting items in the next basket and generating music playlist.

A classic approach for *travel trajectory recommendation* is ranking locations with latent factors learned by collaborative filtering (Zhang and Wang, 2015; Cheng et al., 2013), such approaches suggest top-ranked places but ignore the sequential structure. Another approach regards route recommendation as a planning problem (Brilhante et al., 2013; Gionis et al., 2014; Lim et al., 2015; Anagnostopoulos et al., 2017; Chen et al., 2018; He et al., 2018), which generally assume a fixed objective function that is not directly optimised to predict user behaviour.

The problem of *next basket prediction* suggests the set of next items a user might like to purchase, given the sequence of their shopping basket purchases (Rendle et al., 2010; Wang et al., 2015). The canonical approach for this problem is to apply matrix factorisation to the Markov chain of transitions between items, or modelling high quality transitions using recurrent neural networks (RNNs) (Yu et al., 2016). These methods are feasible because one is only interested in predicting a single element at a time, instead of suggesting an entire sequence given the initial item.

The problem of *music playlist generation* considers recommending a playlist of songs to users, given a query such as a seed song or artist (McFee and Lanckriet, 2011; Chen et al., 2012; Hariri et al., 2012; Bonnin and Jannach, 2014; Hidasi et al., 2015; Ben-Elazar et al., 2017). A canonical approach is to learn a latent representation of songs from historical playlists, and exploit a Markovian assumption on the song transitions; or to learn high quality transitions using RNNs (Choi et al., 2016). We investigate this problem in more detail in Chapter 5.

As a remark, the unary terms in our sequence scoring function (4.2) can be replaced with *personalised* terms to each user, such as from a recommender system (Koren et al., 2009; Rendle et al., 2009). We also note that the trajectory recommendation problem suits RNNs based techniques. However, these techniques generally require much more training data than available in our problem. It is also not clear how to recommend a path (or even sequence) as a whole instead of recommending the next best location as proposed in recent RNNs based work (Liu et al., 2016b). We leave this and personalising path recommendation as future work.

4.6 Summary

We cast the problem of recommending travel trajectories as a special case of path recommendation, which we solve by augmenting the SSVMs. This approach ensures global cohesion of recommended trajectories. The list Viterbi algorithm that was originally invented to decode digital signals corrupted by noise or to find more than one candidate sentence in speech recognition, is employed to account for the existence of multiple ground truths for a given input, as well as recommending trajectories without loops. Experiments on real-world trajectory datasets show that path recommendation approaches outperform existing, non-structured approaches. Our viewpoint also enables researchers to bring recent advances in structured prediction to bear on the problem of recommending structured objects, including further improving the efficiency of inference and learning. In the other direction, techniques from recommender systems to capture latent user- and POI-representations, in sufficiently rich domains, may improve the predictive power of structured prediction models.

Chapter 5

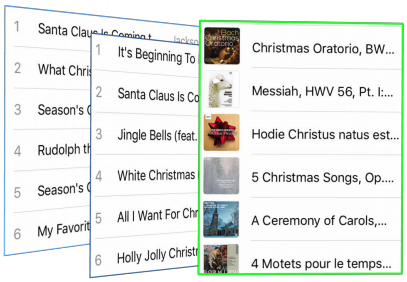
Music Playlist Recommendation with Multi-task Learning

In this chapter, we study the problem of recommending a set of songs from a music library, which is a typical task involved in playlist recommendation. In particular, we investigate this recommendation task in cold-start scenarios (Section 5.2), and propose a set recommendation approach that optimises a bipartite ranking loss which encourages songs in a playlist to be ranked higher than those that are not in it (Section 5.4.3). Further, our approach jointly learns user representations by employing the multi-task learning paradigm (Section 5.4.1), and an equivalence between bipartite ranking and binary classification is exploited for efficient optimisation (Section 5.4.4). Lastly, we present empirical evidences that show the effectiveness of the proposed set recommendation approach (Section 5.5).

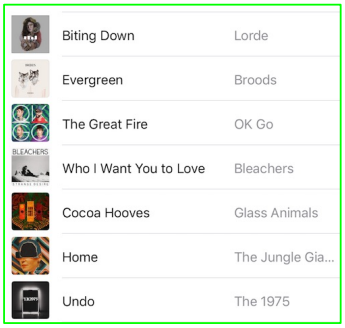
5.1 Introduction

Online music streaming services (e.g., Spotify, Pandora and Apple Music) play an increasingly important role in the digital music industry. A key ingredient of these services is the ability to automatically recommend songs to help users explore large collections of music. Such recommendation is often in the form of a *playlist*, which involves a subset of songs in a large music library.

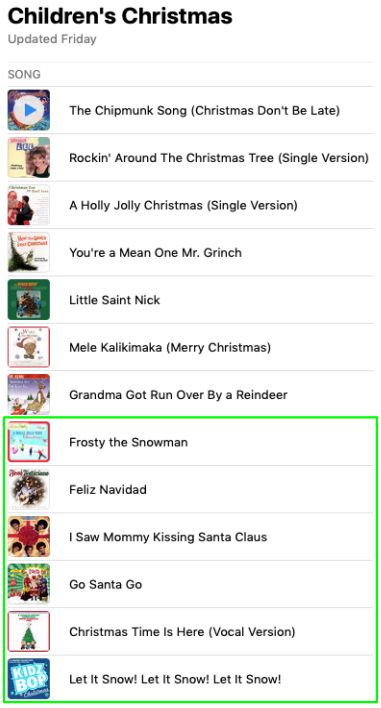
We study the problem of recommending a set of songs to form personalised playlists in *cold-start* scenarios, where there is no historical data for either users or songs. Conventional recommender systems for books or movies (Sarwar et al., 2001; Netflix, 2006) typically learn a score function via matrix factorisation (Koren et al., 2009), and recommend the item that achieves the highest score. This approach is not directly suited to cold-start settings due to the lack of interaction data. Further, in playlist recommendation, one has to recommend a subset of a large collection of



(a) Cold Playlist



(b) Cold User



(c) Cold Song

Figure 5.1: Illustration of the three cold-start settings in playlist recommendation. In each setting, we recommend a set of songs (shown in the green box); **(a) Cold Playlist**: recommending a personalised playlist for a user given her existing playlists; **(b) Cold User**: recommending a playlist for a new user; **(c) Cold Song**: recommending a set of newly released songs to extend an existing playlist.

songs instead of only one top ranked song. Enumerating all possible such subsets is intractable; additionally, it is likely that more than one playlist is satisfactory, since users generally maintain more than one playlist when using a music streaming service, which leads to challenges in standard supervised learning.

5.2 Problem statement

A music playlist is a list of songs that can be listened either in a particular order or in a random order (e.g., shuffle mode). In this thesis, we follow the later case and formulate a music playlist as a set of songs (for further discussion, see Section 5.6). Therefore, the problem of recommending music playlists is an instance of set recommendation (Section 2.1.2). We investigate three cold-start settings in playlist recommendation: *cold playlist*, *cold user* and *cold song*, as illustrated in Figure 5.1. In the *cold playlist* setting, a target user (i.e., the one for whom we recommend playlist) maintains a

number of playlists that can be learned by the recommendation algorithm to suggest a new personalised playlist. In the *cold user* setting, however, we may only know a few simple attributes of a new user (e.g., age, gender and country) or nothing except her user identifier. The recommendation algorithm leverages the playlists from existing users to suggest a new playlist for the new user. In the *cold song* setting, the recommendation algorithm suggests songs from a collection of newly released songs to extend an existing playlist. We assume the algorithm has access to content information and metadata (e.g., artist, genre and audio data) of the newly released songs as well as all playlists from existing users.

Suppose we have a dataset $\mathcal{S}$ with N playlists from U users, where songs in every playlist are from a music collection $\mathcal{C}$ with M songs. Content information and metadata (e.g., artist, genre and audio data) of all songs in $\mathcal{C}$ are available, and we extract a D dimensional vector of features for each song. Depending on the playlist dataset, a few simple attributes of users (e.g., age, gender and country) could be provided, or nothing except the identifiers of users are known. We assume each user has at least one playlist, and each song in the collection $\mathcal{C}$ appears in at least one playlist. Recall that in set recommendation (Section 2.1.2), one learns a function that maps a query to a group of sets. We remark that the target user acts as the query in the *cold playlist* and *cold user* settings, and both the target user and songs in the specified playlist serve as the query in the *cold song* setting. In the *cold playlist* setting, although no contextual information about the new playlist is available besides the target user, her existing playlists could implicitly provide her preference. On the other hand, a new user without any listening history in the *cold user* setting poses more challenges than in the other two settings.

In this thesis, we propose a method to address the problem of recommending playlists in the three cold-start settings. It ranks songs by optimising a bipartite ranking loss (Freund et al., 2003; Agarwal and Niyogi, 2005; Clémençon et al., 2008; Kotlowski et al., 2011), and learns the representations of all users jointly by adopting the multi-task learning paradigm (Section 2.6). We then present how a key equivalence between bipartite ranking and binary classification can make the learning of parameters more efficient. Compared to a straightforward approach that learns a classifier from historical playlists and recommends songs by leveraging the classification results, the proposed approach optimises the top of a ranked list of songs, with a tunable parameter that controls how much emphasis should be put on the top. Observing that the number of songs in a playlist is typically much smaller than the total number of songs in a music library, it makes sense to concentrate only at the top-ranked songs. Before detailing the proposed method (Section 5.4), we review recent work related to the problems and techniques presented in this chapter.

Table 5.1: Glossary of frequently used symbols in this chapter.

Symbol	Meaning	Definition
$\mathcal{S}$	Playlist dataset	§5.2
$\mathcal{C}$	Music library	§5.2
D	Number of features for a song in $\mathcal{C}$	§5.2
M	Number of songs in $\mathcal{C}$	§5.2
M_+^i	Number of songs in playlist i	Eq. (5.12)
M_-^i	Number of songs not in playlist i	Eq. (5.5)
N	Number of playlists in dataset $\mathcal{S}$	§5.2
U	Number of users in dataset $\mathcal{S}$	§5.2
P_u	The set of indices of playlists from user u in dataset $\mathcal{S}$	§5.4.1
$\boldsymbol{\alpha}_u$	Weights of user u, $\boldsymbol{\alpha}_u \in \mathbb{R}^D$	§5.4.1
$\boldsymbol{\beta}_i$	Weights of playlist i, $\boldsymbol{\beta}_i \in \mathbb{R}^D$	§5.4.1
$\boldsymbol{\mu}$	Weights shared by all users (and playlists), $\boldsymbol{\mu} \in \mathbb{R}^D$	§5.4.1
θ	The collection of all parameters: $\{\{\boldsymbol{\alpha}_u\}_{u=1}^U, \{\boldsymbol{\beta}_i\}_{i=1}^N, \boldsymbol{\mu}\}$	§5.4.1
$\mathbf{x}_m$	Feature vector of song m, $\mathbf{x}_m \in \mathbb{R}^D$	§5.4.1
y_m^i	A binary label indicates whether song m is in playlist i	Eq. (5.5)
$\mathbf{w}_{u,i}$	Weights of playlist i from user u, $i \in P_u$, $\mathbf{w}_{u,i} \in \mathbb{R}^D$	Eq. (5.2)
ξ_i	Slack variable for playlist i	Eq. (5.8)
p	Hyper-parameter of multi-task classification, $p \in \mathbb{R}^+$	§5.4.4
$\lambda_1, \lambda_2, \lambda_3$	Regularisation constants	Eq. (5.4)
$f(m,u,i)$	Affinity score between user u, playlist i and song m	Eq. (5.1)
$r_m^{(a)}, r_m^{(b)}, r_m^{(c)}$	Score of song m in cold-start settings	Table 5.2
$R_\theta(f,\mathcal{S})$	Empirical risk of f on dataset $\mathcal{S}$ (a placeholder)	Eq. (5.3)
$\Omega(\theta)$	Regularisation term in the multi-task learning objective	Eq. (5.4)
$\ell(f,u,i)$	Loss of f for playlist i from user u	Eq. (5.5)
$\widetilde{\ell}(f,u,i)$	Another loss of f for playlist i from user u	Eq. (5.11)
$R_\theta^{\text{RANK}}(f,\mathcal{S})$	Empirical risk with loss $\ell(f,u,i)$	Eq. (5.6)
$\widetilde{R}_\theta^{\text{RANK}}(f,\mathcal{S})$	Empirical risk with loss $\widetilde{\ell}(f,u,i)$	Eq. (5.10)
$R_\theta^{\text{MTC}}(f,\mathcal{S})$	Empirical risk of multi-task classification	Eq. (5.12)

We remark that this chapter introduces quite a few symbols, and the frequently used ones can be found in Table 5.1.

5.3 Related work

We summarise recent work most related to playlist recommendation and music recommendation in cold-start scenarios, as well as work on the connections between bipartite ranking and binary classification.

5.3.1 Playlist recommendation

There is a rich set of recent literature on playlist recommendation, which can be summarised into two typical settings: playlist generation and next song recommendation. Playlist generation is to produce a complete playlist given some seed. For example, the AutoDJ system (Platt et al., 2001) generates playlists given one or more seed songs; Groove Radio can produce a personalised playlist for the specified user given a seed artist (Ben-Elazar et al., 2017); or a seed location in hidden space (where all songs are embedded) can be specified in order to generate a complete playlist (Chen et al., 2012). There are also works that focus on evaluating the learned playlist model, without concretely generating playlists (McFee and Lanckriet, 2011, 2012). See this survey (Bonnin and Jannach, 2014) for more details.

Next song recommendation predicts the next song a user might play after observing some context. For example, the most recent sequence of songs with which a user has interacted were used to infer the contextual information, which was then employed to rank the next possible song via a topic-based sequential model learned from users' existing playlists (Hariri et al., 2012). Context can also be the artists in a user's listening history, which has been employed to score the next song together with the frequency of artist collocations as well as song popularity (McFee et al., 2012; Bonnin and Jannach, 2013). It is straightforward to produce a complete playlist using next song recommendation techniques, i.e., by picking the next song sequentially (Bonnin and Jannach, 2013; Ben-Elazar et al., 2017).

5.3.2 Cold-start recommendation

In the collaborative filtering literature, the cold-start setting has primarily been addressed through suitable regularisation of matrix factorisation parameters based on exogenous user- or item-features (Ma et al., 2008; Agarwal and Chen, 2009; Cao et al., 2010). Regularisation techniques for deep neural networks (e.g., dropout) have also been shown to help cold-start recommendation (Volkovs et al., 2017). The novel idea of jointly factorising a document-term matrix and a document-user matrix (Saveski and Mantrach, 2014) achieved promising cold-start document recommendations, especially with regularisation that encourages local smoothness in the learned embeddings.

Content-based approaches can handle the recommendation of new songs, typically by making use of content features of songs extracted either automatically (Seyerlehner et al., 2010; Eghbal-Zadeh et al., 2015) or manually by musical experts (John, 2006). Further, content features can also be combined with other approaches, such as those based on collaborative filtering (Yoshii et al., 2006; Donaldson, 2007; Shao et al., 2009), which is known as the hybrid recommendation approach (Burke, 2002; Aggarwal,

2016). Another popular approach for cold-start recommendation involves explicitly mapping user- or item- content features to latent embeddings (Gantner et al., 2010). This approach can be adopted to recommend new songs, e.g., by learning a convolutional neural network to map audio features of new songs to the corresponding latent embeddings (Oord et al., 2013), which were then used to score songs together with the latent embeddings of playlists. The problem of recommending music for new users can also be tackled using a similar approach, e.g., by learning a mapping from user attributes to user embeddings.

A slightly different approach to deal with music recommendation for new users is learning hierarchical representations for genre, sub-genre and artist. By adopting an additive form with user and artist weights, it can fall back to using only artist weights when recommending music to new users; if the artist weights are not available (e.g., a new artist), this approach further falls back to using the weights of sub-genre or genre (Ben-Elazar et al., 2017). However, the requirement of seed information (e.g., artist, genre or a seed song) restricts its direct applicability to the *cold playlist* and *cold user* settings. Further, encoding song usage information as features makes it unsuitable for recommending new songs directly.

5.3.3 Connections between bipartite ranking and binary classification

It has been established that bipartite ranking and binary classification are closely related (Ertekin and Rudin, 2011; Narasimhan and Agarwal, 2013; Menon and Williamson, 2016). In particular, Ertekin and Rudin (2011) showed that the objective of the P-Norm Push and that of the P-Classification share the same minimiser(s). Further, the P-Norm Push objective is an approximation of the Infinite-Push objective (Agarwal, 2011), or equivalently, the objective of Top-Push (Li et al., 2014), which focuses on the highest ranked negative example instead of the lowest ranked positive example in the Bottom-Push objective (Rudin, 2009) adopted in this work.

Compared to the Bayesian Personalised Ranking (BPR) approach (Rendle et al., 2009; McFee et al., 2012) that requires all positive items to be ranked higher than those unobserved ones, the adopted approach only penalises unobserved items that are ranked higher than the lowest ranked positive item, which can be optimised more efficiently when only the top ranked items are of interest (Rudin, 2009; Li et al., 2014).

5.4 Multi-task learning for playlist recommendation

We first introduce a multi-task learning objective and then show how the playlist recommendation problem can be handled in the three cold-start settings. We discuss the

challenges in optimising the multi-task learning objective via convex constrained optimisation when a bipartite ranking loss is adopted, and show how one can efficiently approximate an optimal solution by minimising an unconstrained objective.

5.4.1 Multi-task learning objective

Let P_u denote the set of (indices of) playlists from user $u \in \{1, \dots, U\}$. We aim to learn a function $f(m, u, i)$ that measures the affinity between song $m \in \{1, \dots, M\}$ and playlist $i \in P_u$ from user u. Given the feature vector $\mathbf{x}_m \in \mathbb{R}^D$ of song m, suppose the affinity function $f : \mathbb{R}^D \to \mathbb{R}$ takes the form of a linear function, i.e.,

$$f(m, u, i) = \mathbf{w}_{u,i}^\top \mathbf{x}_m, \tag{5.1}$$

where $\mathbf{w}_{u,i} \in \mathbb{R}^D$ are the weights of playlist i from user u.

Inspired by Ben-Elazar et al. (2017) where the weights of a playlist are decomposed into user weights and artist weights, we decompose $\mathbf{w}_{u,i}$ into three components[1]

$$\mathbf{w}_{u,i} = \boldsymbol{\alpha}_u + \boldsymbol{\beta}_i + \boldsymbol{\mu}, \tag{5.2}$$

where $\boldsymbol{\alpha}_u \in \mathbb{R}^D$ are weights for user u, $\boldsymbol{\beta}_i \in \mathbb{R}^D$ are weights specific for playlist i, and $\boldsymbol{\mu} \in \mathbb{R}^D$ are the weights shared by all users (and playlists). This decomposition allows us to learn the user weights $\boldsymbol{\alpha}_u$ using all her playlists, and exploit all training playlists when learning the shared weights $\boldsymbol{\mu}$.

Let θ denote all parameters in $\big\{\{\boldsymbol{\alpha}_u\}_{u=1}^U, \{\boldsymbol{\beta}_i\}_{i=1}^N, \boldsymbol{\mu}\big\}$. The learning task is to minimise the empirical risk of affinity function f on dataset $\mathcal{S}$ over parameters θ, i.e.,

$$\min_{\theta} \ \Omega(\theta) + R_\theta(f, \mathcal{S}), \tag{5.3}$$

where $\Omega(\theta)$ is a regularisation term and $R_\theta(f, \mathcal{S})$ denotes the empirical risk of f on $\mathcal{S}$ (Vapnik, 1992), which will be discussed in the later part of this section.

For each user u, $u \in \{1, \dots, U\}$, we can define a machine learning task which recommends playlists for the user by learning from her historical playlists, and the above formulation allows us to jointly learn the parameters from the U tasks. We therefore call the objective in problem (5.3) the *multi-task learning objective*.[2]

We further assume that playlists from the *same* user have *similar* weights and the shared weights $\boldsymbol{\mu}$ are sparse (i.e., users only share a small portion of their weights). To impose these assumptions, we apply ℓ_1 regularisation to encourage sparse parameters

[1] This method is also related to random effects models (Laird et al., 1982; Diggle et al., 2002).

[2] We also adopt a shared representation $\boldsymbol{\mu}$ in the formulation, which is typical in multi-task learning (Caruana, 1997; Ruder, 2017).

Table 5.2: Methods to rank songs in three cold-start settings.

Cold-start Setting	Song Scoring Method
(a) Cold Playlist	$r_m^{(a)} = (\boldsymbol{\alpha}_u + \boldsymbol{\mu})^\top \mathbf{x}_m$
(b) Cold User	$r_m^{(b)} = \left(\frac{1}{\lvert\mathcal{U}\rvert} \sum_{u \in \mathcal{U}} \boldsymbol{\alpha}_u + \boldsymbol{\mu} \right)^\top \mathbf{x}_m$ or $r_m^{(b)} = \boldsymbol{\mu}^\top \mathbf{x}_m$
(c) Cold Song	$r_m^{(c)} = (\boldsymbol{\alpha}_u + \boldsymbol{\beta}_i + \boldsymbol{\mu})^\top \mathbf{x}_m$

in $\boldsymbol{\beta}_i$ and $\boldsymbol{\mu}$. The regularisation term in our multi-task learning objective is

$$\Omega(\theta) = \lambda_1 \sum_{u=1}^{U} \|\boldsymbol{\alpha}_u\|_2^2 + \lambda_2 \sum_{i=1}^{N} \|\boldsymbol{\beta}_i\|_1 + \lambda_3 \|\boldsymbol{\mu}\|_1, \tag{5.4}$$

where constants $\lambda_1, \lambda_2, \lambda_3 \in \mathbb{R}^+$, and the ℓ_2 regularisation term is to penalise large values in user weights. We specify the empirical risk $R_\theta(f, \mathcal{S})$ in Section 5.4.3.

5.4.2 Cold-start playlist recommendation

Once parameters θ have been learned, we make a recommendation by first scoring each song according to available information (e.g., an existing user or playlist), then form or extend a playlist by either taking the set of top-K scored songs or sampling a set of songs with probabilities proportional to their scores. Specifically, in the *cold playlist* setting where the target user u is known, we score song m as $r_m^{(a)} = (\boldsymbol{\alpha}_u + \boldsymbol{\mu})^\top \mathbf{x}_m$.

Further, in the *cold user* setting where simple attributes of the new user are available, we first approximate the weights of the new user using the average weights of similar existing users (i.e., users whose attribute vectors are similar to that of the new user in terms of e.g., the cosine similarity), then we can score song m as $r_m^{(b)} = \left(\frac{1}{\lvert\mathcal{U}\rvert} \sum_{u \in \mathcal{U}} \boldsymbol{\alpha}_u + \boldsymbol{\mu} \right)^\top \mathbf{x}_m$, where $\mathcal{U}$ is the set of (e.g., 10) existing users that are most similar to the new user. On the other hand, if we know nothing about the new user except her identifier, we can simply score song m as $r_m^{(b)} = \boldsymbol{\mu}^\top \mathbf{x}_m$ where $\boldsymbol{\mu}$ is the shared weights that can be interpreted as a prior from a Bayesian point of view (Ben-Elazar et al., 2017).

Lastly, in the *cold song* setting where we are given a specific playlist i from user u, we therefore can score song m using both user weights and playlist weights, i.e., $r_m^{(c)} = (\boldsymbol{\alpha}_u + \boldsymbol{\beta}_i + \boldsymbol{\mu})^\top \mathbf{x}_m$. Table 5.2 summaries the methods presented above for ranking songs in the three cold-start settings.

We now specify the empirical risk $R_\theta(f, \mathcal{S})$ in problem (5.3) and develop methods to efficiently optimise the multi-task learning objective.

5.4.3 Ranking songs via Bottom-Push

We aim to rank songs that are likely to be in a playlist above those that are unlikely to be chosen when making a recommendation. To achieve this, we optimise the multi-task learning objective by minimising a bipartite ranking loss. In particular, we minimise the number of songs that are not in a training playlist but are ranked above the lowest ranked song in it. This is known as the Bottom-Push objective (Rudin, 2009), and the penalty of the affinity function f for playlist i from user u is

$$\ell(f,u,i) = \frac{1}{M^i_-} \sum_{m':y^i_{m'}=0} \left[\!\!\left[\min_{m:y^i_m=1} f(m,u,i) \le f(m',u,i) \right]\!\!\right], \tag{5.5}$$

where M^i_- is the number of songs not in playlist i, binary variable y^i_m denotes whether song m appears in playlist i, and indicator function $[\![\cdot]\!]$ represents the $0/1$ loss.

The empirical risk when employing the bipartite ranking loss $\ell(f,u,i)$ is

$$R^{\text{RANK}}_\theta(f,\mathcal{S}) = \frac{1}{N} \sum_{u=1}^{U} \sum_{i\in P_u} \ell(f,u,i). \tag{5.6}$$

There are two challenges to optimise the multi-task learning objective with the empirical risk R^{RANK}_θ, namely, the non-differentiable $0/1$ loss and the *min* function in $\ell(f,u,i)$. To address these challenges, we first upper-bound the $0/1$ loss with one of its convex surrogates, e.g., the exponential loss $[\![z \le 0]\!] \le e^{-z}$,

$$\ell(f,u,i) \le \frac{1}{M^i_-} \sum_{m':y^i_{m'}=0} \exp\left(f(m',u,i) - \min_{m:y^i_m=1} f(m,u,i) \right). \tag{5.7}$$

One approach to deal with the *min* function in $\ell(f,u,i)$ is introducing slack variables ξ_i to lower-bound the scores of songs in playlist i and transform problem (5.3) with empirical risk R^{RANK}_θ into a convex constrained optimisation problem

$$\begin{aligned} \min_\theta \ & \Omega(\theta) + \frac{1}{N} \sum_{u=1}^{U} \sum_{i\in P_u} \frac{1}{M^i_-} \sum_{m':y^i_{m'}=0} \exp\left(f(m',u,i) - \xi_i\right) \\ s.t. \ & \xi_i \le f(m,u,i), \\ & u \in \{1,\dots,U\},\ i \in P_u,\ m \in \{1,\dots,M\} \cap \{m : y^i_m = 1\}. \end{aligned} \tag{5.8}$$

Note that the number of constraints in problem (5.8) is $\sum_{u=1}^{U} \sum_{i\in P_u} \sum_{m=1}^{M} [\![y^i_m = 1]\!]$, i.e., the accumulated playcount of all songs, which is of order $O(\bar{L}N)$ asymptotically, where $\bar{L}$ is the average number of songs in playlists (typically less than 100). However, the total number of playlists N can be enormous in production systems (e.g.,

Spotify hosts more than 2 billion playlists (Spotify, 2018)) which imposes a significant challenge in optimisation. This issue could be alleviated by applying the cutting-plane method (Avriel, 2003) or the sub-gradient method. Unfortunately, we find both methods converge extremely slowly for this problem in practice. In particular, the cutting-plane method is required to solve a constrained optimisation problem with at least N constraints in each iteration, which remains challenging.

5.4.4 Efficient optimisation

Alternatively, we can approximate the *min* function in $\ell(f, u, i)$ using the well known Log-sum-exp function (Boyd and Vandenberghe, 2004), i.e.,

$$\min_j z_j = -\max_j(-z_j) = -\lim_{p\to+\infty} \frac{1}{p} \log \sum_j \exp(-pz_j), \tag{5.9}$$

which allows us to approximate the empirical risk R_θ^{RANK} (with the exponential surrogate) by $\widetilde{R}_\theta^{\text{RANK}}$ defined as

$$\widetilde{R}_\theta^{\text{RANK}}(f, \mathcal{S}) = \frac{1}{N} \sum_{u=1}^{U} \sum_{i\in P_u} \frac{1}{M_-^i} \left[\widetilde{\ell}(f, u, i)\right]^{\frac{1}{p}}, \tag{5.10}$$

where hyper-parameter $p \in \mathbb{R}^+$ and

$$\widetilde{\ell}(f, u, i) = \sum_{m:y_m^i=1} \left[\sum_{m':y_{m'}^i=0} \exp(-(f(m, u, i) - f(m', u, i)))\right]^p. \tag{5.11}$$

We further observe that $\widetilde{\ell}(f, u, i)$ can be transformed into the objective of the standard P-Norm Push (Rudin, 2009) by simply swapping the positives $\{m : y_m^i = 1\}$ and negatives $\{m' : y_{m'}^i = 0\}$. Inspired by the connections between bipartite ranking and binary classification (Ertekin and Rudin, 2011; Narasimhan and Agarwal, 2013; Menon and Williamson, 2016), we swap the positives and negatives in the objective of the P-Classification (Ertekin and Rudin, 2011) while taking care of signs. This results in an empirical risk with a classification loss:

$$R_\theta^{\text{MTC}}(f, \mathcal{S}) = \frac{1}{N} \sum_{u=1}^{U} \sum_{i\in P_u} \left(\frac{1}{pM_+^i} \sum_{m:y_m^i=1} \exp(-pf(m, u, i)) + \frac{1}{M_-^i} \sum_{m':y_{m'}^i=0} \exp(f(m', u, i))\right), \tag{5.12}$$

where M_+^i is the number of songs in playlist i.

Lemma 1. *Let* $\theta^* \in \operatorname{argmin}_\theta R_\theta^{\text{MTC}}$ *(assuming minimisers exist), then* $\theta^* \in \operatorname{argmin}_\theta \widetilde{R}_\theta^{\text{RANK}}$.

Proof. See Appendix E for a complete proof. Alternatively, we can use the proof of the equivalence between the P-Norm Push and the P-Classification (Ertekin and Rudin, 2011) if we swap the positives and negatives in R_θ^{MTC} and $\widetilde{R}_\theta^{\text{RANK}}$. □

By Lemma 1, we can optimise the parameters of the multi-task learning objective by solving a (convex) unconstrained optimisation problem:[3]

$$\min_\theta \ \Omega(\theta) + R_\theta^{\text{MTC}}(f, \mathcal{S}). \tag{5.13}$$

Problem (5.13) can be efficiently optimised using the Orthant-Wise Limited-memory Quasi-Newton (OWL-QN) algorithm (Andrew and Gao, 2007), an L-BFGS variant that can address ℓ_1 regularisation effectively. We refer to the approach that solves problem (5.13) as *Multi-task Classification* (MTC). As a remark, optimal solutions of problem (5.13) are not necessarily the optimal solutions of $\min_\theta \ \Omega(\theta) + \widetilde{R}_\theta^{\text{RANK}}$ due to regularisation. However, when parameters θ are small (which is generally the case when using regularisation), optimal solutions of the two objectives can nonetheless approximate each other well.

5.5 Experiments

We present empirical evaluations for cold-start playlist recommendation on two real playlist datasets, and compare the proposed multi-task classification approach to a number of well known baseline approaches.

5.5.1 Dataset

We evaluate on two publicly available playlist datasets: the 30Music (Turrin et al., 2015) and the AotM-2011 (McFee and Lanckriet, 2012) datasets. The Million Song Dataset (MSD) (Bertin-Mahieux et al., 2011) serves as an underlying dataset where songs in all playlists are intersected (i.e., filtering out songs not in the MSD); additionally, song and artist information in the MSD are used to compute song features.

30Music Dataset is a collection of listening events and user-generated playlists retrieved from Last.fm.[4] We first intersect the playlists data with songs in the MSD, then filter out playlists with less than 5 songs. This results in about 17K playlists over 45K songs from 8K users.

[3] We choose not to directly optimise the empirical risk $\widetilde{R}_\theta^{\text{RANK}}$, which involves the objective of P-Norm Push, since classification loss can be optimised more efficiently here (Ertekin and Rudin, 2011).

[4] https://www.last.fm

Table 5.3: Statistics of music playlist datasets.

	30Music	AotM-2011
Playlists	17,457	84,710
Users	8,070	14,182
Avg. Playlists per User	2.2	6.0
Songs	45,468	114,428
Avg. Songs per Playlist	16.3	10.1
Artists	9,981	15,698
Avg. Artists per Playlist	11.5	9.0
Avg. Songs per Artist	4.6	7.1

AotM-2011 Dataset is a collection of playlists shared by Art of the Mix[5] users during the period from 1998 to 2011. Songs in playlists have been matched to those in the MSD. It contains roughly 84K playlists over 114K songs from 14K users after filtering out playlists with less than 5 songs.

Table 5.3 summarises the two playlist datasets used in this work. The histograms of the number of playlists per user as well as song popularity (i.e., the accumulated playcount of the song in the training set) of the two datasets are shown in Figure 5.2 and Figure 5.3, respectively. We can see that both the number of playlists per user (Figure 5.2) and song popularity (Figure 5.3) follow a long-tailed distribution, which imposes further challenge to the learning task as the amount of data is very limited for users (or songs) at the tail.

5.5.2 Features

Song metadata, audio data, genre and artist information, as well as song popularity and artist popularity (i.e., the accumulated playcount of all songs from the artist in the training set) are encoded as features. The metadata of songs (e.g., duration, year of release) and pre-computed audio features (e.g., loudness, mode, tempo) are from the MSD. We use genre data from the Top-MAGD genre dataset (Schindler et al., 2012) and tagtraum genre annotations for the MSD (Schreiber, 2015) via one-hot encoding. If the genre of a song is unknown, we apply mean imputation using genre counts of songs in the training set. To encode artist information as song features, we create a sequence of artist identifiers for each playlist in the training set, and train a word2vec model (Mikolov et al., 2013) that learns embeddings of artists. We also assume no popularity information is available for newly released songs, and therefore song popularity is not a feature in the *cold song* setting. Finally, we add a constant feature (with value 1.0) for each song to account for bias.

[5]http://www.artofthemix.org

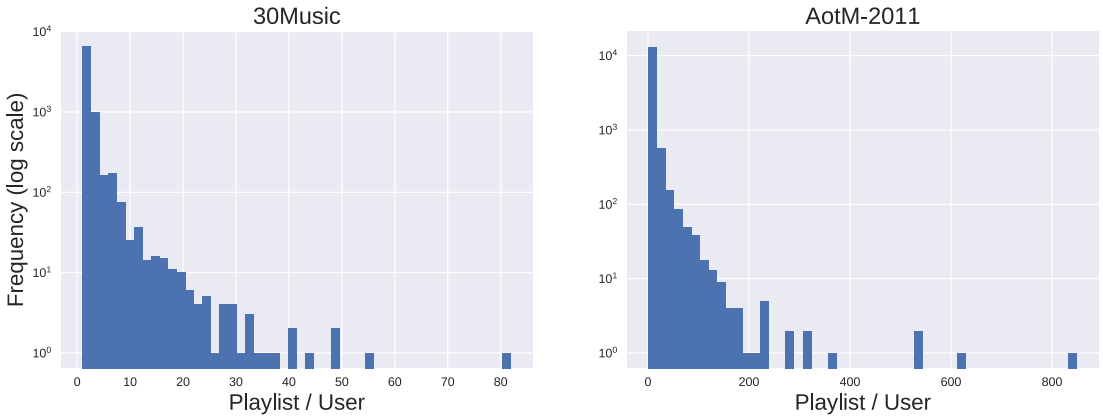


Figure 5.2: Histogram of the number of playlists per user.

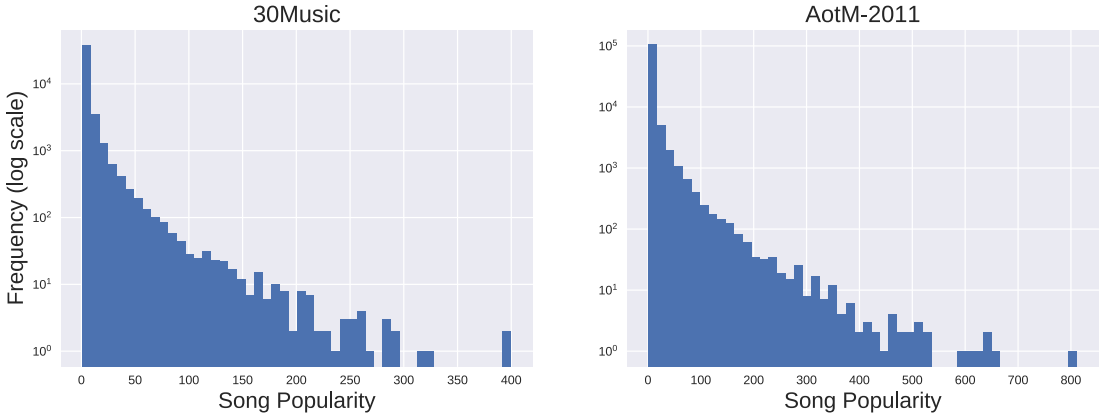


Figure 5.3: Histogram of song popularity.

5.5.3 Experimental setup

In each of the three cold-start settings, we first split the two playlist datasets into training and test sets, then evaluate the performance of the proposed method (on the test set), and compare it against several baseline approaches.

Dataset split In the *cold playlist* setting, we hold a portion of the playlists from about 20% of users in both datasets for testing, and all other playlists are used for training. The test set is formed by sampling playlists where each song has been included in at least four other playlists among the whole dataset. We also make sure each song in the test set appears in the training set, and all users in the test set have a few playlists in the training set. In the *cold user* setting, we sample 30% of users and hold all of their playlists for testing in both datasets. Similarly, we require songs in the test set to appear in the training set, and a user will thus not be used for testing if holding all of her playlists breaks this requirement. Lastly, in the *cold song* setting, we hold 5K of

Table 5.4: Statistics of datasets in three cold-start settings.

		Training Set			Test Set		
		Users	Playlists	Songs	Users	Playlists	Songs
Cold Playlist	30Music	8,070	15,262	45,468	1,644	2,195	45,468
	AotM-2011	14,182	75,477	114,428	2,722	9,233	114,428
Cold User	30Music	5,649	14,067	45,468	2,420	3,390	45,468
	AotM-2011	9,928	76,450	114,428	4,254	8,260	114,428
Cold Song	30Music	8,034	17,342	40,468	8,034	8,215	5,000
	AotM-2011	14,177	84,646	104,428	14,177	19,504	10,000

the latest released songs in the 30Music dataset, and 10K of the latest released songs in the AotM-2011 dataset where more songs are available. We remove a playlist from the training set if all songs in it have been held for testing.

Table 5.4 summaries the dataset splits in three cold-start settings.

Baselines We compare the performance of our proposed method (i.e., MTC) with the following baseline approaches in each of the three cold-start settings:

- The *Popularity Ranking* (PopRank) method scores a song using only its popularity in the training set. In the *cold song* setting where song popularity is not available, a song is scored by the popularity of the corresponding artist.
- The *Same Artists - Greatest Hits* (SAGH) (McFee et al., 2012) method scores a song by its popularity if the artist of the song appears in the given user's playlists (in the training set); otherwise the song is scored zero. In the *cold song* setting, this method only considers songs from artists that appear in the given playlist, and scores a song using the popularity of the corresponding artist.
- The *Collocated Artists - Greatest Hits* (CAGH) (Bonnin and Jannach, 2013) method is a variant of SAGH. It scores a song using its popularity, but weighted by the frequency of the collocation between the artist of the song and artists that appear in the given user's playlists (in the training set). In the *cold user* setting, we use the 10 most popular artists instead of artists in the user's listening history (which is not available), and the *cold song* setting is addressed in the same way as in SAGH (i.e., considering only those artists that appear in the given playlist).

We also compare with a variant of Matrix Factorisation (MF) in each setting, which first learns the latent factors of songs, playlists or users, then scores each song by the dot product of the corresponding latent factors. Recommendations are made as per the proposed method. To be specific,

- In the *cold playlist* setting, we factorise the song-user playcount matrix using the weighted matrix factorisation (WMF) algorithm (Hu et al., 2008), which learns the latent factors of songs and users. We call this method WMF.
- In the *cold user* setting, we first learn the latent factors of songs and users by factorising the song-user playcount matrix using WMF, then approximate the latent factors of a new user by the average latent factors of the k (e.g., 100) nearest neighbours (in terms of the cosine similarity between user attribute vectors) in the training set. We call this method WMF+kNN.[6]
- In the *cold song* setting, we factorise the song-playlist matrix to learn the latent factors of songs and playlists, which are further employed to train a fully-connected neural network that maps the content features of a song to its corresponding latent factors (Gantner et al., 2010; Oord et al., 2013). We can then obtain the latent factors of a new song as long as its content features are available. We call this method MF+MLP.

Evaluation We first evaluate all approaches using two accuracy metrics that have been widely adopted in playlist recommendation tasks: HitRate@K and Area under the ROC curve (AUC).

HitRate@K (Hariri et al., 2012), which is also known as Recall@K (Jurafsky and Martin, 2009), is the number of correctly recommended songs amongst the top-K recommendations over the number of songs in the observed playlist. It has been widely employed to evaluate playlist generation and next song recommendation methods (Hariri et al., 2012; Bonnin and Jannach, 2013, 2014; Jannach et al., 2015).

Area under the ROC curve (AUC) (Manning et al., 2008), which is the probability that a positive instance is ranked higher than a negative instance (on average). AUC has been primarily used to measure the performance of classifiers. It has been applied to evaluate playlist generation methods when the task has been cast as a sequence of classification problems (Ben-Elazar et al., 2017).

It is believed that useful recommendations need to include previously unknown items, and this ability can be measured by *Novelty* (Herlocker et al., 2004; Zhang et al., 2012; Schedl et al., 2017),

$$\text{Novelty@K} = \frac{1}{U}\sum_{u=1}^{U}\frac{1}{|P_u^{\text{test}}|}\sum_{i\in P_u^{\text{test}}}\sum_{m\in S_K^i}\frac{-\log_2 pop_m}{K}, \tag{5.14}$$

where P_u^{test} is the (indices of) test playlists from user u, S_K^i is the set of top-K recommendations for test playlist i and pop_m is the popularity of song m. Intuitively, the more popular a song is, the more likely a user is to be familiar with it, and therefore the less likely to be novel.

[6]The WMF+kNN method does not apply to the AotM-2011 dataset in the cold user setting, since such user attributes (e.g., age, gender and country) are not available in the dataset.

We also adopt another beyond-accuracy metric called *Spread* (Kluver and Konstan, 2014), which measures the ability of a recommender system to spread its attention across all possible items. It is defined as the entropy of the distribution of all songs,

$$\text{Spread} = -\sum_{m=1}^{M} \mathbb{P}(m) \log \mathbb{P}(m), \tag{5.15}$$

where $\mathbb{P}(m)$ denotes the probability of song m being recommended, which is computed from the scores of all possible songs using the *softmax* function in this work.

Novelty and Spread are two of the beyond-accuracy metrics that are specifically tailored to recommender systems. Unlike the AUC and Hit Rate, where higher values indicate better performance, here *moderate* values are usually preferable for these two beyond-accuracy metrics (Kluver and Konstan, 2014; Schedl et al., 2017). However, when comparing the performance of various recommendation methods in terms of Novelty and Spread, it is generally unclear which one achieves a moderate value, and this is unfortunately a reflection of current literature (Schedl et al., 2017).

To make these beyond-accuracy metrics consistent with the accuracy metrics employed in this work (i.e., higher values indicate better performance), we observe that the lower- and upper-bound of both Novelty and Spread can be obtained straightforwardly. If an optimal value can be further specified, we can map the values of these two metrics into scores in range $[0, 1]$ such that the optimal value is mapped to 1, and other values are mapped to lower scores that could reflect how far away they are from the optimal value. Let v_L and v_U be the lower- and upper-bound, respectively. One example of such transformation is the following non-linear function

$$f(v) = \begin{cases} 0, & v \in \{v_L, v_U\} \\ 2\sigma\left(\gamma(v)(v - v^*)\right), & v \in (v_L, v_U) \end{cases} \tag{5.16}$$

where $\sigma(x) = (1 + e^{-x})^{-1}$ is the sigmoid function, $v^* \in (v_L, v_U)$ is the specified optimal value, function

$$\gamma(v) = \begin{cases} C(v^* - v_L)^{-1}, & v \le v^* \\ C(v^* - v_U)^{-1}, & v > v^* \end{cases} \tag{5.17}$$

and constant $C \in \mathbb{R}^+$ is a scaling parameter. Figure 5.4 illustrates that function $f(\cdot)$ can map values uniformly distributed on the horizontal axis to scores in $[0, 1]$, with a higher resolution near the optimal value.

In experiment, we specify[7] that $v^* = (v_U + v_L)/2$ and $C = (v_U - v_L)/2$, which results in $f(v) = 2\sigma(-|v - v^*|)$ when $v \in (v_L, v_U)$. Note that PopRank scores songs by their popularity, which, by definition (5.14), will achieve the lowest Novelty. In contrast, the highest Novelty can be obtained by recommending the set of least popular songs, which we call LeastPop in Figure 5.6. In particular, let nov_K^{PopRank}

[7] If additional domain information is available, we can choose the values of v^* and C accordingly.

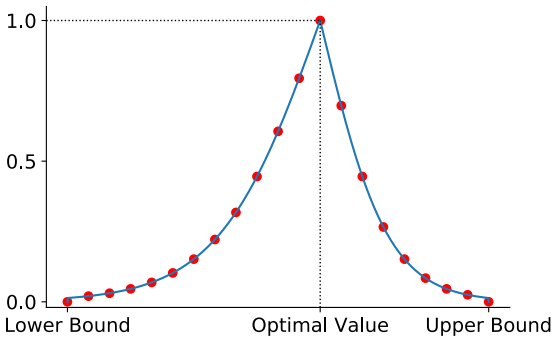


Figure 5.4: Illustration of a non-linear transformation for Novelty and Spread. A non-linear function that maps a set of real numbers into scores in $[0, 1]$ such that a moderate value is mapped to a high score and low/high values to low scores.

Table 5.5: AUC for playlist recommendation in three cold-start settings. *Higher* values indicate better performance.

Cold Playlist			Cold User			Cold Song		
Method	30Music	AotM-2011	Method	30Music	AotM-2011	Method	30Music	AotM-2011
PopRank	94.0	93.8	PopRank	88.3	**91.8**	PopRank	70.9	76.5
CAGH	94.8	94.2	CAGH	86.3	88.1	CAGH	68.0	77.4
SAGH	64.5	79.8	SAGH	54.5	53.7	SAGH	51.5	53.6
WMF	79.5	85.4	WMF+kNN	84.9	N/A	MF+MLP	81.4	80.8
MTC	**95.9**	**95.4**	MTC	**88.8**	**91.8**	MTC	**86.6**	**84.3**

and nov_K^{LeastPop} denote the Novelty@K of PopRank and LeastPop, respectively. The transformation for Novelty@K is $f^{\text{Novelty}}(nov_K^{\text{PopRank}}) = f^{\text{Novelty}}(nov_K^{\text{LeastPop}}) = 0$, and

$$f^{\text{Novelty}}(nov_K) = 2\sigma\left(-\left|nov_K - nov_K^*\right|\right), \quad nov_K \in (nov_K^{\text{PopRank}}, nov_K^{\text{LeastPop}}) \tag{5.18}$$

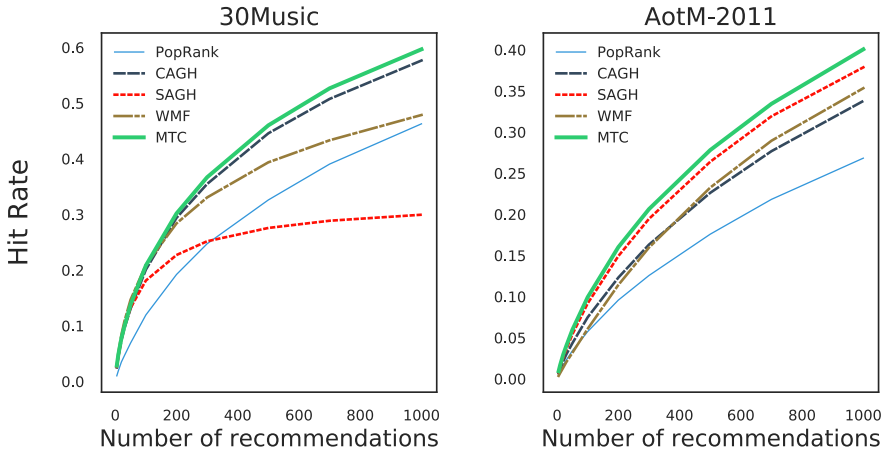
where we specify the optimal value $nov_K^* = \frac{1}{2}\left(nov_K^{\text{PopRank}} + nov_K^{\text{LeastPop}}\right)$. In addition, we know from the definition (5.15) that Spread $s \in [0, \log_2 M]$, where M is the number of songs in consideration. Therefore, by specifying the optimal value $s^* = (\log_2 M)/2$, the Spread can be transformed using $f^{\text{Spread}}(0) = f^{\text{Spread}}(\log_2 M) = 0$, and

$$f^{\text{Spread}}(s) = 2\sigma(-|s - s^*|), \quad s \in (0, \log_2 M). \tag{5.19}$$

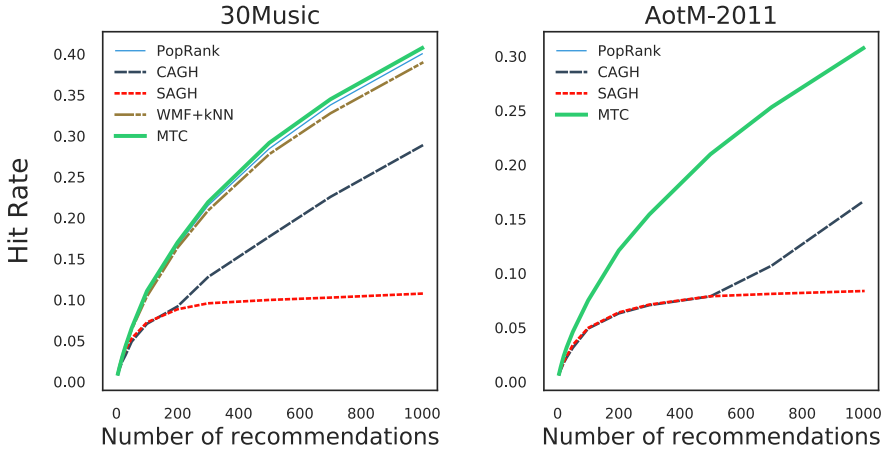
5.5.4 Results

We analyse the empirical results of the proposed method (i.e., MTC) as well as many baselines in terms of both accuracy metrics (i.e., HitRate and AUC) and beyond accuracy metrics (i.e., Novelty and Spread).

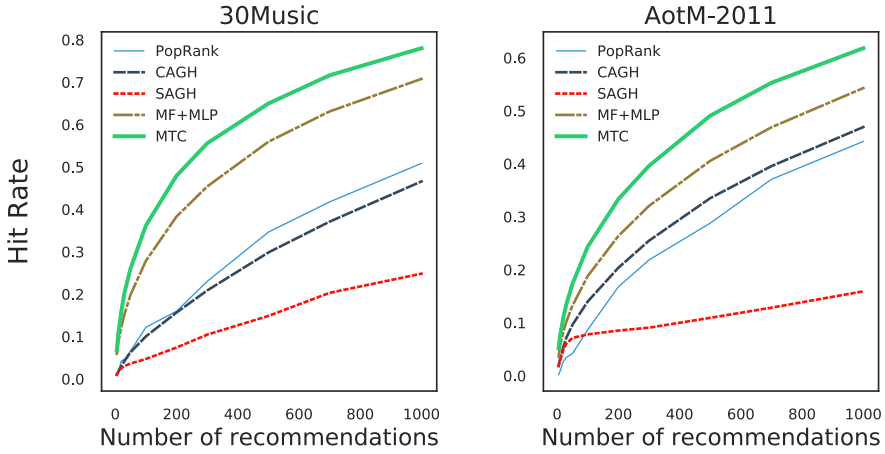
Accuracy Table 5.5 shows the performance of all methods in terms of AUC. We can see that PopRank achieves good performance in all three cold-start settings. This is in line with results reported in (Bonnin and Jannach, 2013, 2014). Artist information,



(a) Cold Playlist



(b) Cold User



(c) Cold Song

Figure 5.5: Hit Rate of playlist recommendation in three cold-start settings. *Higher* values indicate better performance.

particularly the frequency of artist collocations that is exploited in CAGH, improves recommendation in the *cold playlist* and *cold song* settings. Further, PopRank is one of the best performing methods in the *cold user* setting, which is consistent with previous discoveries (McFee et al., 2012; Bonnin and Jannach, 2013, 2014). The reason is believed to be the long-tailed distribution of songs in playlists (Cremonesi et al., 2010; Bonnin and Jannach, 2013). The MF variant does not perform well in the *cold playlist* setting, but it performs reasonably well in the *cold user* setting when attributes of new users are available (i.e., in the 30Music dataset), and it works particularly well in the *cold song* setting where both song metadata and audio features of new songs are provided.

Lastly, MTC is the best performing method in all three cold-start settings on both datasets. Interestingly, it is the tied best on the AotM-2011 dataset in the *cold user* setting (recall that this dataset does not provide user attributes such as age, gender and country), and it achieves the same performance as PopRank in the *cold user* setting on the AotM-2011 dataset, which suggests that MTC might degenerate to simply ranking songs according to the popularity when making recommendations for new users; however, when simple attributes of new users are available, it can improve the recommendations by exploiting information learned from existing users.

Figure 5.5 shows the Hit Rate of all methods in three cold-start settings when the number of recommended songs K varies from 5 to 1000. As expected, the performance of all methods improves when the number of recommendations increases. We can see from Figure 5.5a that SAGH and CAGH perform better than PopRank (except for SAGH on the 30Music dataset when K is larger than 300) in the *cold playlist* setting, which confirms that artist information is helpful in retrieving songs in ground truth playlists (i.e., improving recall). It is interesting to observe that the performance of WMF is always between SAGH and CAGH on both datasets, although the performance of both SAGH and CAGH vary significantly across datasets. This might suggest that this variant of matrix factorisation is more robust than approaches based on ranking according to song popularity and artist information.

It is challenging to improve upon simply ranking by song popularity in the *cold user* setting, as shown in Figure 5.5b, which is in line with previous discoveries (McFee et al., 2012; Bonnin and Jannach, 2013, 2014). In contrast, learning-based approaches (i.e., MTC and MF+MLP) always perform better than other baselines that use only artist information in the *cold song* setting (Figure 5.5c). PopRank works surprisingly well; it even outperforms CAGH which exploits artist collocations on the 30Music dataset. The fact that CAGH always performs better than SAGH confirms that artist collocation is helpful for music recommendation.

In summary, MTC outperforms all other methods by a big margin on both datasets in the *cold song* setting (Figure 5.5c). It performs as well as PopRank in the *cold user*

Table 5.6: *Raw* Spread for playlist recommendation in three cold-start settings. *Moderate* values are preferable.

Cold Playlist			Cold User			Cold Song		
Method	30Music	AotM-2011	Method	30Music	AotM-2011	Method	30Music	AotM-2011
PopRank	9.8	10.5	PopRank	9.8	10.5	PopRank	7.4	7.8
CAGH	5.8	2.3	CAGH	4.2	5.3	CAGH	4.3	4.6
SAGH	10.3	10.4	SAGH	10.0	10.7	SAGH	6.5	5.9
WMF	10.7	11.6	WMF+kNN	10.7	N/A	MF+MLP	8.5	9.2
MTC	9.4	10.4	MTC	9.9	11.4	MTC	7.9	8.3

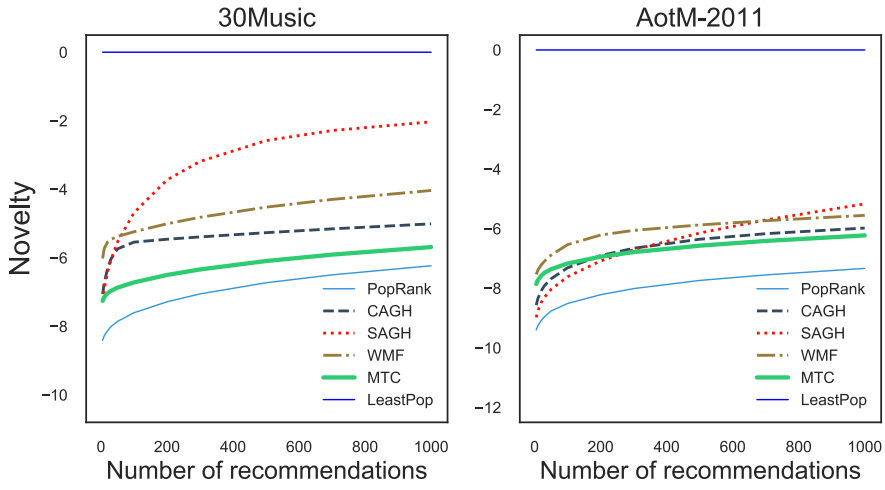
Table 5.7: *Transformed* Spread for playlist recommendation in three cold-start settings. *Higher* values indicate better performance.

Cold Playlist			Cold User			Cold Song		
Method	30Music	AotM-2011	Method	30Music	AotM-2011	Method	30Music	AotM-2011
PopRank	.225	.219	PopRank	**.225**	**.219**	PopRank	.833	.708
CAGH	.252	.004	CAGH	.057	.086	CAGH	.062	.044
SAGH	.143	**.239**	SAGH	.188	.183	SAGH	.450	.151
WMF	.098	.078	WMF+kNN	.098	N/A	MF+MLP	.636	.621
MTC	**.319**	**.239**	MTC	.206	.095	MTC	**.918**	**.949**

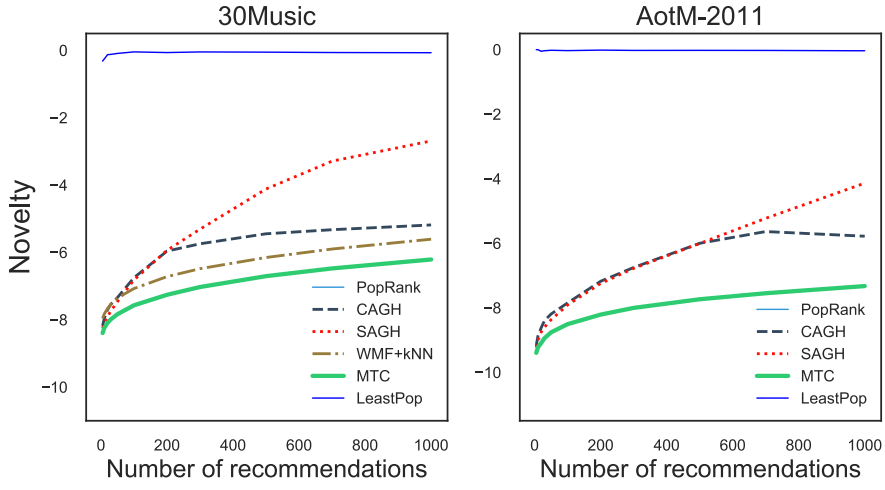
setting; however, MTC can improve the recommendations when simple attributes of new users are available (Figure 5.5b). We also observe that MTC outperforms other baselines in the *cold playlist* setting (Figure 5.5a), although the margin is not as big as that in the *cold song* setting. This demonstrates the effectiveness of the proposed approach for cold-start playlist recommendation.

Beyond accuracy Table 5.6 shows the performance of all recommendation approaches in terms of *Spread*. In the *cold song* setting, CAGH and SAGH focus on songs from artists in users' listening history (and similar artists), which explains the relative low *Spread*. However, in the *cold playlist* and *cold user* settings, SAGH improves its attention spreading due to the set of songs it focuses on is significantly bigger (i.e., songs from all artists in users' previous playlists and songs from the 10 most popular artists, respectively). Surprisingly, CAGH remains focusing on a relatively small set of songs in both settings. Lastly, in all three cold-start settings, the MF variants have the highest *Spread*, while both PopRank and MTC have (similar) moderate *Spread*.

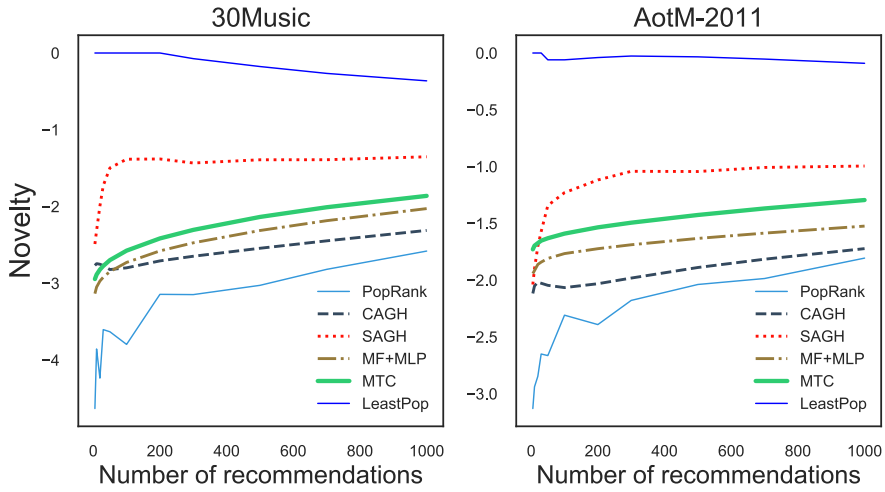
Table 5.7 summaries the *Spread* of all methods after the non-linear transformation (5.19). We can see that PopRank achieves very good performance in all three cold-start settings, in particular, it is the best method in the *cold user* setting. Further, MTC performs the best in both the *cold playlist* and *cold song* settings. These observations are consistent with the results in Table 5.6 which indicates PopRank and MTC generally perform better than other methods in terms of *Spread*.



(a) Cold Playlist



(b) Cold User



(c) Cold Song

Figure 5.6: *Raw* Novelty of playlist recommendation in three cold-start settings. *Moderate* values are prefereable.

Table 5.8: The mean of *transformed* Novelty for playlist recommendation in three cold-start settings. *Higher* values indicate better performance.

Cold Playlist			Cold User			Cold Song		
Method	30Music	AotM-2011	Method	30Music	AotM-2011	Method	30Music	AotM-2011
PopRank	.000	.000	PopRank	.000	.000	PopRank	.000	.000
CAGH	.234	.102	CAGH	.131	.103	CAGH	.591	.648
SAGH	**.454**	.117	SAGH	**.294**	**.161**	SAGH	**.890**	**.923**
WMF	.413	**.184**	WMF+kNN	.088	N/A	MF+MLP	.594	.750
MTC	.109	.108	MTC	.053	.031	MTC	.664	.845

Figure 5.6 shows the *Novelty* of all methods in three cold-start settings. The values of *Novelty* of all methods raise as the number of recommendations increases (except those of LeastPop which we shall discuss later). We can see from Figure 5.6a that PopRank has the lowest *Novelty* in the *cold playlist* setting, which is not surprising given its definition (Equation 5.14). Both SAGH and CAGH start with low *Novelty* and grow as the number of recommended songs increases, but the *Novelty* of CAGH saturates much earlier than that of SAGH. The reason could be that, when the number of recommendations is larger than the total number of songs from artists in a user's existing playlists, SAGH will simply recommend songs randomly (which are likely to be novel) while CAGH will recommend songs from artists that are similar to those in the user's existing playlists (which could be comparably less novel). Further, MTC achieves lower *Novelty* than WMF and CAGH, which indicates that MTC tends to recommend popular songs to form new playlists.

It is interesting to observe that MTC and PopRank perform identically in the *cold user* setting, as shown in Figure 5.6b. SAGH has the largest *Novelty* on both datasets, likely for similar reasons to those in the *cold playlist* setting. The performance of different methods (in terms of *Novelty*) in the *cold song* setting (Figure 5.6c) are similar to those in the *cold playlist* setting (Figure 5.6b); however, we note that there are two differences: (i) The *Novelty* of SAGH saturates after the number of recommendations reaches a certain value (roughly 60), the reason could be that, on average, the total number of songs from the set of artists in a playlist is about 60, as shown in Table 5.3; (ii) MTC achieves higher *Novelty* than both WMF and CAGH, which might suggest that MTC tends to recommend new songs that will be (comparably) less popular.

Table 5.8 shows the transformed *Novelty* averaged over different values of K (i.e., the number of recommendations). The transformed *Novelty* of both PopRank and LeastPop should be 0 by definition (5.18), and we observe that SAGH achieves the overall highest transformed *Novelty* in all settings (except in the *cold playlist* setting on the AotM-2011 dataset where the MF variant performs best). This is consistent with the results in Figure 5.6, where the raw *Novelty* of SAGH is generally "moderate"

considering that the lower- and upper-bound of *Novelty* are attained by PopRank and LeastPop respectively. In addition, CAGH achieves decent performance in all three cold-start settings. Interestingly, although MTC and the MF variants attain respectable performance in both the *cold playlist* and *cold song* settings (on at least one dataset), they do not perform well in the *cold user* setting in terms of transformed *Novelty*.

To conclude, for the task of recommending playlists in cold-start scenarios, the proposed method MTC performs best in terms of accuracy metrics Hit Rate and AUC. Despite being outperformed by a comparatively simpler method SAGH in terms of the beyond-accuracy metric *Novelty*, MTC produces the overall best recommendations in terms of another beyond-accuracy metric *Spread*.

5.6 Discussion

We discuss the relationship between multi-task learning, bipartite ranking and binary classification (from the perspective of loss function). In addition, we also remark several design choices adopted in this work and compare our problem setup with a closely related setting considered in a recent RecSys challenge.

5.6.1 Multi-task learning, bipartite ranking and binary classification

Multi-task learning is a method that learns more than one tasks in parallel by using a shared representation to achieve inductive transfer between tasks, it could improve generalisation accuracy of a particular task by leveraging additional signals from related tasks (Caruana, 1993, 1997). Sharing representation among multiple tasks allows us to jointly learn the parameters of users and playlists, as well as the shared parameters from multiple recommendation tasks, which further enables us to deal with the problem of recommending a set of songs in three cold-start settings.

The bipartite ranking loss adopted in this work (i.e., the Bottom-Push) guides the learning process such that the learned parameters will (generally) reflect our intention to rank the set of songs in a playlist higher than those that are not in it. Ideally, we can directly optimise this loss function using training data; unfortunately, this is infeasible due to the enormous number of constraints in the involved optimisation problem, we therefore resort to minimise an approximation of the constrained objective. It turns out that the approximation transforms the Bottom-Push objective to (a variant of) the P-Norm Push objective. Although one can optimise the objective of the P-Norm Push using standard techniques (e.g., gradient descent), more efficient optimisation can be achieved if we make use of an equivalence between bipartite ranking and binary classification, which results in an unconstrained objective with a classification loss.

5.6.2 Cold-start playlist recommendation versus playlist continuation

In the *cold playlist* setting, we recommend more than one playlists for a given user; however, all these recommendations are identical as the ranking of songs for a specific user is the same (Table 5.2). This is due to the fact that no other contextual information for a recommendation is available except the user identity. Similarly, in the *cold user* setting, the same set of songs will always be suggested no matter how many times the recommendation have been made because of the lack of contextual information in each recommendation. A more plausible and perhaps more realistic setting is to provide one or more *seed* songs for each task,[8] and the recommended playlist should be cohesive with the given seed. This setup is known as *playlist continuation* (Schedl et al., 2017), which has been explored in a recent RecSys challenge.[9] One may notice that the setup of playlist continuation is similar to the *cold song* setting, except that the set of songs to be added to a playlist are not necessarily newly released songs.

5.6.3 Information of songs, playlists and users

In this thesis, we assume that content features of songs (e.g., metadata, audio data) are provided, even for newly released songs. On the other hand, no user (or playlist) feature is available. We may have a few simple attributes of users (e.g., age, gender and country) or only user identifiers are known. In practice, users might reveal their preferences in profiles, and playlist metadata (e.g., title, description and created time) might also be available, which could be leveraged by the learning algorithm.

Further, we treat a playlist as a set of songs by discarding the sequential order. It turns out that the sequential order of songs in a playlist has not been well understood (Schedl et al., 2017), some work suggest that the order of songs and song-to-song transitions are important for the quality of the recommended playlist (McFee and Lanckriet, 2012; Kamehkhosh et al., 2018), while other work discover that the order of songs seems to be negligible, but the ensemble (i.e., set) of songs in a playlist do matter (Tintarev et al., 2017; Vall et al., 2017).

As a remark, in the *cold user* setting, we approximate the weights (or latent factors) of a new user using the average weights (or latent factors) of similar users in the training set in MTC (or WMF+kNN). One could also use a weighted average (e.g., weighted by the normalised cosine similarity between user attribute vectors) of those similar users' weights (or latent factors), however, we did not find any significant difference in performance compared to the arithmetic mean in the experiments.

[8] The seed information can also be an artist or a genre, as considered in Ben-Elazar et al. (2017).
[9] http://www.recsyschallenge.com/2018/

5.7 Summary

We study the problem of recommending a set of songs to form playlists in three cold-start settings: *cold playlist*, *cold user* and *cold song*. We adopt the multi-task learning paradigm that learns user- and playlist-specific weights as well as shared weights from user-curated playlists, which allows us to form new personalised playlists for an existing user, and to produce playlists for a new user, or to extend users' playlists with newly released songs. We optimise the parameters by minimising a bipartite ranking loss that ranks the set of songs in a playlist above songs that are not in it. An equivalence between bipartite ranking and binary classification further enables efficient approximation of optimal parameters. Empirical evaluations on two real playlist datasets demonstrate the effectiveness of the proposed method for cold-start playlist recommendation. For future work, we would like to explore auxiliary data sources (e.g., music information shared on social media) and additional features of songs and users (e.g., lyrics, user profiles) to make better recommendations.

Chapter 6

Conclusion

In this chapter, we first summarise the work presented in previous chapters, then discuss the limitations of the proposed approaches for recommending structured objects. We end this chapter by presenting a few potential directions for future work.

6.1 Research summary

Efficient recommendation of structured objects has been a challenge for recommender system research. We developed techniques for recommending structured objects, in particular, for *paths* and *sets*.

We studied the problem of recommending travel trajectories, an instance of path recommendation, in Chapter 3 and Chapter 4. Learning to rank and route planning techniques have been utilised to combine POI preferences and transition patterns in Chapter 3. Our experiments on trajectory datasets in five cities suggested that both POI preferences and transition patterns were helpful in recommending trajectories, and taking advantage of both information sources results in better recommendations.

Chapter 4 continued the study of this problem but from a structured prediction perspective. We extended the structured support vector machines framework and proposed a structured recommendation approach for suggesting paths. A list variant of the classic Viterbi algorithm has been employed for efficient learning and recommendation. Empirically evaluation demonstrated the proposed structured recommendation approach achieved improvement over previous approaches.

We investigated an example of set recommendation in Chapter 5, i.e., the problem of recommending a set of songs from a music library to form a new playlist or extend an existing playlist. A bipartite ranking loss was adopted to encourage songs in a playlist to be ranked above those that are not in it. We employed the multi-task learning paradigm to jointly learn the representations of multiple users, and leveraged an equivalence between bipartite ranking and binary classification to achieve efficient learning. We compared the proposed set recommendation approach with many

well-known playlist recommendation methods, and experimental results on two real playlist datasets suggested our approach based on multi-task learning and bipartite ranking can recommend better playlists.

Overall, the work on path and set recommendation in this thesis demonstrates the promising potential of employing machine learning and planning techniques for recommendation tasks.

6.2 Future work

There are a few limitations of our work on recommending structured objects, which could motivate a number of potential research work in this area. In particular, the underlying model of our structured recommendation approach is a sequence model based on the Markov assumption, which can be violated in many practical scenarios. Addressing these limitations could result in better path recommendation techniques that can be more widely useful in practice.

One of the fundamental techniques for set recommendation is bipartite ranking, in particular, a loss function that encourages any element in the desired set to be ranked above the highest ranked undesired element. Relying on a single highest ranked element could result in a recommendation method that is fragile to noise. Since it is highly likely that data records contain noise in practice, methods that are robust to noise in data are highly desired. On the other hand, the set recommendation techniques could be helpful to path recommendation, e.g., a two-stage approach that first fixes the set of elements then determines a particular order to form a path.

At last, there are many other types of important structures besides path and set, for example, the *tree* structure used in natural language processing (Jurafsky and Martin, 2009) and evolutionary genetics analysis (Charlesworth and Charlesworth, 2010); the *graph* (i.e., network) structure widely employed in representing interactions between entities in biological and technological systems (Newman, 2010). The tasks of recommending such structures would likely to bring both challenges and opportunities to the study of recommendation systems.

Appendix A

Cutting-plane Methods

The goal of cutting-plane methods is to find or localise a point in a convex *target set* $Z \in \mathbb{R}^n$, or determine that Z is empty in some cases. The method does not assume any direct access to the description of Z, such as the objective or constraint functions in an optimisation problem, except through a *cutting-plane oracle*.

The cutting-plane methods work by generating a query point q and pass it to the oracle, the oracle either tells us that $q \in Z$ (in which case we are done), or it returns a hyperplane which separates q from Z. This hyperplane is called a *cutting-plane*, or *cut*, since it eliminates a half-space from our search, as illustrated in Figure A.1. The procedure is repeated until we find a point in Z or determine that Z is empty. Cutting-plane methods are also known as *localisation* methods (Boyd and Vandenberghe, 2008). Algorithm 5 shows a conceptual description of cutting-plane methods.

A.1 Overview of cutting-plane methods

Consider a convex optimisation problem with M constraints,

$$\begin{aligned} &\min_z \ f_0(z) \\ &s.t. \ \ f_i(z) \leq 0, \quad i \in \{1, \ldots, M\} \end{aligned} \tag{A.1}$$

where $f_0, \ldots, f_M$ are convex and differentiable, and the target set Z is the optimal (or ε-suboptimal) set. For a query point q, the oracle first checks for its feasibility. If q is not feasible, this means that at least one constraint in problem (A.1) is violated. Suppose constraint $f_j(z) \leq 0$ is violated by q, then we have $f_j(q) > 0$. In addition, as $f_j(z)$ is convex and differentiable, we have the following inequality[1]

$$f_j(z) \geq f_j(q) + \nabla f_j(q)^\top (z - q), \tag{A.2}$$

[1] Intuitively, the RHS of (A.2) is a tangent hyperplane at point q, and a convex function lies above its tangent hyperplanes.

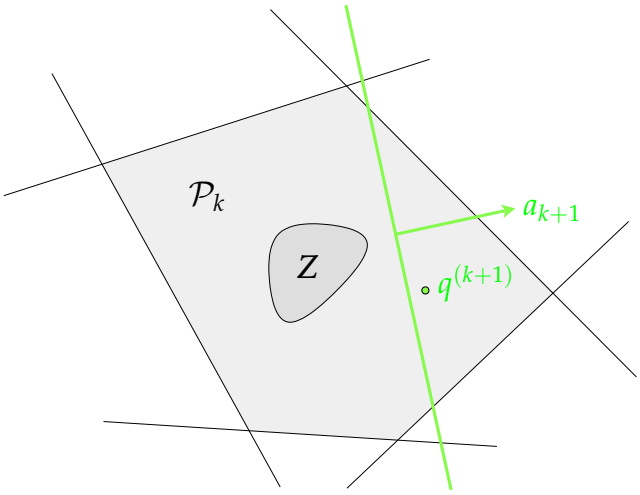


Figure A.1: Illustration of cutting-plane methods. The inequality (i.e., cutting-plane) $a_{k+1}^\top z \leq b_{k+1}$ separates the query point $q^{(k+1)}$ in polyhedron $\mathcal{P}_k$ from the target set Z.

and we conclude that if $f_j(q) + \nabla f_j(q)^\top (z - q) > 0$, then $f_j(z) > 0$, which violated the constraint $f_j(z) \leq 0$ in problem (A.1). Thus, any feasible point should satisfy the following inequality

$$f_j(q) + \nabla f_j(q)^\top (z - q) \leq 0. \tag{A.3}$$

Equation (A.3) is called a *feasibility cut* for problem (A.1), since it cuts away the half-space $\{z \mid f_j(q) + \nabla f_j(q)^\top (z - q) > 0\}$ with infeasible points. If more than one constraint is violated by q, we can generate a *feasibility cut* for each violated constraint.

On the other hand, if q is feasible, and suppose $\nabla f_0(q) \neq 0$ (otherwise q is optimal and we are done), by the inequality (A.2) we have

$$f_0(z) > f_0(q), \text{ if } \nabla f_0(q)^\top (z - q) > 0.$$

In other words, any point that satisfies inequality $\nabla f_0(q)^\top (z - q) > 0$ has an objective value larger than $f_0(q)$ and hence cannot be optimal. We can exclude these points by forming the following cutting-plane

$$\nabla f_0(q)^\top (z - q) \leq 0, \tag{A.4}$$

which is called an *objective cut* for problem (A.1), and it cuts out the half-space $\{z \mid \nabla f_0(q)^\top (z - q) > 0\}$ with non-optimal points.

Further, if we keep track of the best objective value for all feasible query points

$$f_{\text{best}} = f_0(q_{\text{best}}) = \min\{f_0(q^{(t)}) \mid q^{(t)} \text{ is feasible}\},$$

Algorithm 5 The cutting-plane algorithm

Given: an initial polyhedron $\mathcal{P}_0$ that contains target set Z.
$k = 0$
repeat
  Generate a query point $q^{(k+1)}$ in $\mathcal{P}_k$
  Query the oracle at $q^{(k+1)}$
  if The oracle determines that $q^{(k+1)} \in Z$ **then**
    return $q^{(k+1)}$
  else if The oracle returns a cutting-plane $a_{k+1}^\top z \leq b_{k+1}$ **then**
    Update constraints: $\mathcal{P}_{k+1} = \mathcal{P}_k \cap \{z \mid a_{k+1}^\top z \leq b_{k+1}\}$
  end if
  $k = k + 1$
until Convergence or $\mathcal{P}_{k+1} = \emptyset$

since the optimal point has an objective value which is at most f_{best}, we can cut away the half-space of points $\{z \mid f_0(z) > f_{\text{best}}\}$ with an additional (convex) constraint $f_0(z) \leq f_{\text{best}}$. By inequality (A.2) we have

$$f_0(q) + \nabla f_0(q)^\top (z - q) - f_{\text{best}} \leq 0, \tag{A.5}$$

which is known as a *deep objective cut* (Boyd and Vandenberghe, 2008). Here q is a feasible query point, if $q = q_{\text{best}}$, the cut (A.5) reduces to the objective cut (A.4).

As a remark, for non-differentiable objective or constraints, the gradients $\nabla f_j(z)$, $j \in \{0, \ldots, M\}$ can generally be replaced with sub-gradients.

A.2 Methods to generate query points

Recall that in cutting-plane methods, we would like to generate a query point $q^{(k+1)}$ in the current polyhedron $\mathcal{P}_k$ such that the cut returned by the oracle can reduce the size of $\mathcal{P}_{k+1}$ as much as possible. However, when we query the oracle at point $q^{(k+1)}$, we do not know in which direction of the generated cut that will be excluded from the current polyhedron $\mathcal{P}_k$. If we measure the informativeness of the k-th cut using the volume reduction ratio $\frac{V(\mathcal{P}_{k+1})}{V(\mathcal{P}_k)}$, we seek a point $q^{(k+1)}$ such that, no matter which direction to cut, we can obtain a certain guaranteed volume reduction. Here we describe four typical methods to generate query points, i.e., the method of Kelley-Cheney-Goldstein, the Chebyshev centre method, the analytic centre method, and the centre of gravity method.

A.2.1 Method of Kelley-Cheney-Goldstein

Given query points $q^{(1)}, \ldots, q^{(k)}$, the *method of Kelley-Cheney-Goldstein* generates the next query point $q^{(k+1)}$ by solving a linear program (LP) (Wulff and Ong, 2013):

$$\begin{aligned} &\min_{z,\theta} \ \theta \\ &s.t. \ \ \theta \geq f_0(q^{(i)}) + \nabla f_0(q^{(i)})^\top (z - q^{(i)}), \ \forall i \leq k \\ &\quad\ A_k^\top z \leq \mathbf{b}_k, \end{aligned} \tag{A.6}$$

where $A_k^\top z \leq \mathbf{b}_k$ are the set of constraints that define polyhedron $\mathcal{P}_k$.

Intuitively, the method of Kelley-Cheney-Goldstein greedily chooses the vertex of the current polyhedron $\mathcal{P}_k$ that maximises the convex objective $f_0(z)$ as the next query point. To see this, let

$$t_i(z) = f_0(q^{(i)}) + \nabla f_0(q^{(i)})^\top (z - q^{(i)}),$$

then $t_i(z)$ is a hyperplane tangent to $f_0(z)$ at point $q^{(i)}$. We can rewrite LP (A.6) as

$$\begin{aligned} &\min_{z,\theta} \ \theta \\ &s.t. \ \ \theta \geq \max_{z \in \mathcal{P}_k} \ t_i(z), \ i \in \{1, \ldots, k\}. \end{aligned}$$

Further, let $z_i^* = \operatorname*{argmax}_{z \in \mathcal{P}_k} \ t_i(z), \ i \in \{1, \ldots, k\}$, we can see that z_i^* is either a vertex or a point lies on an edge of polyhedron $\mathcal{P}_k$ (this can be shown intuitively when $\mathcal{P}_k$ is a 2-dimensional region), and the optimal solution of LP (A.6) is $z^* = \operatorname{argmax}_i t_i(z_i^*)$, it follows that the next query point $q^{(k+1)} = z^*$ is either a vertex or a point lies on an edge of $\mathcal{P}_k$. In fact, if we solve LP (A.6) using the simplex algorithm, the optimal solution is guaranteed to be a vertex of $\mathcal{P}_k$.

A.2.2 Chebyshev centre method

If we rescale the gradients $\nabla f_0(q^{(i)})$ to unit length in (A.6), it results in finding the centre of the largest Euclidean ball that lies inside the current polyhedron $\mathcal{P}_k$ (Elzinga and Moore, 1975), in other words, we find the next query point $q^{(k+1)}$ by solving

$$\begin{aligned} &\min_{z,\theta} \ \theta \\ &s.t. \ \ \theta \geq f_0(q^{(i)}) + \frac{\nabla f_0(q^{(i)})}{\|\nabla f_0(q^{(i)})\|}^\top \left(z - q^{(i)}\right), \ \forall i \leq k \\ &\quad\ A_k^\top z \leq \mathbf{b}_k. \end{aligned} \tag{A.7}$$

This variant is called the *Chebyshev centre* method, which has been shown to possess significantly better convergence properties than the method of Kelley-Cheney-Goldstein (Goffin and Vial, 2002).

A.2.3 Analytic centre cutting-plane method

Given a linear constraint $a_i^\top z \leq b_i$, we define a slack variable $s_i = b_i - a_i^\top z$, that is, s_i measures how far the current solution is from the constraint. The *analytic centre* is defined as the unique maximiser of the function $\prod_i s_i$ (Wulff and Ong, 2013), i.e.,

$$\operatorname*{argmax}_z \prod_i s_i = \operatorname*{argmax}_z \sum_{i=1}^{k} \log(b_i - a_i^\top z) + \sum_{j=1}^{M'} \log(d_j - c_j^\top z), \tag{A.8}$$

where we use M' linear constraints to represent the initial polyhedron $\mathcal{P}_0$ and explicitly write them as $c_j^\top z \leq d_j$, $j \in \{1, \ldots, M'\}$. The unique maximiser of (A.8) can be efficiently found using Newton iterations (Goffin and Vial, 2002).

Let polyhedron

$$\mathcal{P}_k = \{z \mid c_j^\top z \leq d_j,\ j \in \{1, \ldots, M'\} \text{ and } a_i^\top z \leq b_i,\ i \in \{1, \ldots, k\}\},$$

the *analytic centre cutting plane method* (ACCPM) chooses the analytic centre of $\mathcal{P}_k$ to query the oracle, which is a good trade-off between simplicity and practical performance (Boyd and Vandenberghe, 2008).

A.2.4 Centre of gravity or Bayes point method

Assume set $\mathcal{C} \subseteq \mathbb{R}^n$ is bounded and has non-empty interior. The centre of gravity (CG) of $\mathcal{C}$ is defined as

$$\mathbf{cg}(\mathcal{C}) = \frac{\int_{\mathcal{C}} z dz}{\int_{\mathcal{C}} dz}. \tag{A.9}$$

The centre of gravity method chooses the point $q^{(k+1)} = \mathbf{cg}(\mathcal{P}_k)$ to query the oracle (Louche and Ralaivola, 2015). It turns out that this method has a very good convergence property in terms of the worst-case volume reduction factor, in particular, we always have

$$\frac{V(\mathcal{P}_{k+1})}{V(\mathcal{P}_k)} \leq 1 - \frac{1}{e} \approx 0.63, \tag{A.10}$$

in other words, the volume of the localisation polyhedron is reduced by at least 37% at each iteration, and this guarantee is completely independent of all problem parameters, including the dimension n. However, it is *extremely difficult* to compute the centre of gravity of a polyhedron in $\mathbb{R}^n$ that is described by a set of linear inequalities,

which makes this method impractical. Variants that compute an approximate centre of gravity have been developed, and some of these approximations can be used to create a practical CG method (Boyd and Vandenberghe, 2008).

Appendix B

Linking Losses for Bipartite Ranking and Binary Classification

We generalise the results in Ertekin and Rudin (2011) and show a parametric family of bipartite ranking losses share minimiser(s) with a parametric family of binary classification losses. Let function $f : \mathcal{X} \to \mathbb{R}$ be

$$f(\mathbf{x}; \mathbf{w}, b) := g(\mathbf{x}; \mathbf{w}) + b, \tag{B.1}$$

where $\mathbf{x}$ is an input vector, $\mathbf{w}$ is a parameter vector, b is a bias parameter, and real-valued function $g(\mathbf{x}; \mathbf{w})$ is bounded and is differentiable with respect to $\mathbf{w}$.

Let $\mathcal{D} = \mathcal{S}_+ \cup \mathcal{S}_-$ denote a binary dataset with a set of positive instances $\mathcal{S}_+$ as well as a set of negative instances $\mathcal{S}_-$. Suppose $\alpha, \beta, \gamma, C \in \mathbb{R}^+$ are *finite* positive numbers, we define R^{BR} be the following parametric family of bipartite ranking risk:

$$R^{\text{BR}}(f, \mathcal{D}) = \left[\sum_{\mathbf{x}_+ \in \mathcal{S}_+} \left[\sum_{\mathbf{x}_- \in \mathcal{S}_-} e^{-\beta(f(\mathbf{x}_+; \mathbf{w}, b) - f(\mathbf{x}_-; \mathbf{w}, b))} \right]^{\frac{\alpha}{\beta}} \right]^{\gamma}. \tag{B.2}$$

Note that R^{BR} is independent of b, since $f(\mathbf{x}_+; \mathbf{w}, b) - f(\mathbf{x}_-; \mathbf{w}, b) = g(\mathbf{x}_+; \mathbf{w}) - g(\mathbf{x}_-; \mathbf{w})$.

Further, let R^{BC} be the following parametric family of binary classification risk:

$$R^{\text{BC}}(f, \mathcal{D}) = \frac{1}{\alpha} \sum_{\mathbf{x}_+ \in \mathcal{S}_+} e^{-\alpha f(\mathbf{x}_+; \mathbf{w}, b)} + \frac{C}{\beta} \sum_{\mathbf{x}_- \in \mathcal{S}_-} e^{\beta f(\mathbf{x}_-; \mathbf{w}, b)}. \tag{B.3}$$

Theorem 1 presents the relation between R^{BR} and R^{BC}.

Theorem 1. *Suppose minimiser(s) of the two risks $R^{\text{BR}}(f, \mathcal{D})$ and $R^{\text{BC}}(f, \mathcal{D})$ exist, given $\mathbf{w}^*$ and $b^* = \frac{1}{\alpha + \beta} \left(\ln \sum_{\mathbf{x}_+ \in \mathcal{S}_+} \exp(-\alpha g(\mathbf{x}_+; \mathbf{w}^*)) - \ln \sum_{\mathbf{x}_- \in \mathcal{S}_-} \exp(\beta g(\mathbf{x}_-; \mathbf{w}^*)) - \ln C \right)$, if $\mathbf{w}^* \in \operatorname{argmin}_{\mathbf{w}} R^{\text{BR}}(f, \mathcal{D})$, then $(\mathbf{w}^*, b^*) \in \operatorname{argmin}_{\mathbf{w}, b} R^{\text{BC}}(f, \mathcal{D})$. In addition, if $(\mathbf{w}^*, \tilde{b}) \in$*

$\operatorname{argmin}_{\mathbf{w},b} R^{\text{BC}}(f,\mathcal{D})$, *then* $\mathbf{w}^* \in \operatorname{argmin}_{\mathbf{w}} R^{\text{BR}}(f,\mathcal{D})$ *and* $\tilde{b} = b^*$.

Proof. Let $s(\mathbf{w}) = \sum_{\mathbf{x}_+ \in \mathcal{S}_+} \exp(-\alpha g(\mathbf{x}_+;\mathbf{w}))$ and $t(\mathbf{w}) = \sum_{\mathbf{x}_- \in \mathcal{S}_-} \exp(\beta g(\mathbf{x}_-;\mathbf{w}))$, we note that $R^{\text{BR}}(f,\mathcal{D}) = [s(\mathbf{w})]^{\gamma} [t(\mathbf{w})]^{\frac{\alpha\gamma}{\beta}}$, and $b^* = \frac{1}{\alpha+\beta} \ln \frac{s(\mathbf{w}^*)}{t(\mathbf{w}^*)C}$.

First, we compute derivatives of R^{BR} and R^{BC} with respect to the parameters.

$$\begin{aligned}\frac{\partial R^{\text{BC}}}{\partial b} &= \frac{-\alpha}{\alpha} \sum_{\mathbf{x}_+ \in \mathcal{S}_+} \exp(-\alpha(g(\mathbf{x}_+;\mathbf{w}) + b)) + \frac{\beta C}{\beta} \sum_{\mathbf{x}_- \in \mathcal{S}_-} \exp(\beta(g(\mathbf{x}_-;\mathbf{w}) + b)) \\ &= -\exp(-\alpha b) s(\mathbf{w}) + C \exp(\beta b) t(\mathbf{w}).\end{aligned} \tag{B.4}$$

$$\begin{aligned}\frac{\partial R^{\text{BR}}}{\partial w_k} &= \gamma [s(\mathbf{w})]^{\gamma-1} \left[\sum_{\mathbf{x}_+ \in \mathcal{S}_+} \exp(-\alpha g(\mathbf{x}_+;\mathbf{w}))(-\alpha) \frac{\partial g(\mathbf{x}_+;\mathbf{w})}{\partial w_k} \right] [t(\mathbf{w})]^{\frac{\alpha\gamma}{\beta}} \\ &\quad + [s(\mathbf{w})]^{\gamma} \frac{\alpha\gamma}{\beta} [t(\mathbf{w})]^{\frac{\alpha\gamma}{\beta}-1} \left[\sum_{\mathbf{x}_- \in \mathcal{S}_-} \exp(\beta g(\mathbf{x}_-;\mathbf{w})) \beta \frac{\partial g(\mathbf{x}_-;\mathbf{w})}{\partial w_k} \right] \\ &= \alpha\gamma [s(\mathbf{w})]^{\gamma-1} [t(\mathbf{w})]^{\frac{\alpha\gamma}{\beta}-1} \left[-t(\mathbf{w}) \sum_{\mathbf{x}_+ \in \mathcal{S}_+} \exp(-\alpha g(\mathbf{x}_+;\mathbf{w})) \frac{\partial g(\mathbf{x}_+;\mathbf{w})}{\partial w_k} \right. \\ &\qquad \left. + s(\mathbf{w}) \sum_{\mathbf{x}_- \in \mathcal{S}_-} \exp(\beta g(\mathbf{x}_-;\mathbf{w})) \frac{\partial g(\mathbf{x}_-;\mathbf{w})}{\partial w_k} \right], \\ &\quad \forall k \in \{1,\dots,d\}.\end{aligned} \tag{B.5}$$

$$\begin{aligned}\frac{\partial R^{\text{BC}}}{\partial w_k} &= \frac{-\alpha}{\alpha} \sum_{\mathbf{x}_+ \in \mathcal{S}_+} \exp(-\alpha(g(\mathbf{x}_+;\mathbf{w}) + b)) \frac{\partial g(\mathbf{x}_+;\mathbf{w})}{\partial w_k} \\ &\quad + \frac{\beta C}{\beta} \sum_{\mathbf{x}_- \in \mathcal{S}_-} \exp(\beta(g(\mathbf{x}_-;\mathbf{w}) + b)) \frac{\partial g(\mathbf{x}_-;\mathbf{w})}{\partial w_k} \\ &= -\exp(-\alpha b) \sum_{\mathbf{x}_+ \in \mathcal{S}_+} \exp(-\alpha g(\mathbf{x}_+;\mathbf{w})) \frac{\partial g(\mathbf{x}_+;\mathbf{w})}{\partial w_k} \\ &\quad + C \exp(\beta b) \sum_{\mathbf{x}_- \in \mathcal{S}_-} \exp(\beta g(\mathbf{x}_-;\mathbf{w})) \frac{\partial g(\mathbf{x}_-;\mathbf{w})}{\partial w_k} \\ &= -\exp(-\alpha b) \left[\sum_{\mathbf{x}_+ \in \mathcal{S}_+} \exp(-\alpha g(\mathbf{x}_+;\mathbf{w})) \frac{\partial g(\mathbf{x}_+;\mathbf{w})}{\partial w_k} \right. \\ &\qquad \left. - C \exp((\alpha+\beta) b) \sum_{\mathbf{x}_- \in \mathcal{S}_-} \exp(\beta g(\mathbf{x}_-;\mathbf{w})) \frac{\partial g(\mathbf{x}_-;\mathbf{w})}{\partial w_k} \right] \\ &\quad \forall k \in \{1,\dots,d\}.\end{aligned} \tag{B.6}$$

Let $\mathbf{w}^* \in \operatorname{argmin}_{\mathbf{w}} R^{\text{BR}}$, then $0 = \frac{\partial R^{\text{BR}}}{\partial w_k}\Big|_{\mathbf{w}=\mathbf{w}^*}$, $\forall k \in \{1, \dots, d\}$, by Eq. (B.5), we have

$$\begin{aligned} & t(\mathbf{w}) \sum_{\mathbf{x}_+ \in \mathcal{S}_+} \exp(-\alpha g(\mathbf{x}_+; \mathbf{w})) \frac{\partial g(\mathbf{x}_+; \mathbf{w})}{\partial w_k}\Big|_{\mathbf{w}=\mathbf{w}^*} \\ & = s(\mathbf{w}) \sum_{\mathbf{x}_- \in \mathcal{S}_-} \exp(\beta g(\mathbf{x}_-; \mathbf{w})) \frac{\partial g(\mathbf{x}_-; \mathbf{w})}{\partial w_k}\Big|_{\mathbf{w}=\mathbf{w}^*}, \ \forall k \in \{1, \dots, d\}. \end{aligned} \tag{B.7}$$

Note that $s(\mathbf{w}^*) = C \exp((\alpha + \beta) b^*) t(\mathbf{w}^*)$, by Eq. (B.4) we have

$$\begin{aligned} \frac{\partial R^{\text{BC}}}{\partial b}\Big|_{\mathbf{w}=\mathbf{w}^*, b=b^*} & = -\exp(-\alpha b^*) s(\mathbf{w}^*) + C \exp(\beta b^*) t(\mathbf{w}^*) \\ & = -\exp(-\alpha b^*) C \exp((\alpha + \beta) b^*) t(\mathbf{w}^*) + C \exp(\beta b^*) t(\mathbf{w}^*) \\ & = 0. \end{aligned} \tag{B.8}$$

Further, note that $\exp((\alpha + \beta) b^*) = \frac{s(\mathbf{w}^*)}{t(\mathbf{w}^*) C}$, by Eq. (B.7) we have

$$\begin{aligned} & C \exp((\alpha + \beta) b^*) \sum_{\mathbf{x}_- \in \mathcal{S}_-} \exp(\beta g(\mathbf{x}_-; \mathbf{w})) \frac{\partial g(\mathbf{x}_-; \mathbf{w})}{\partial w_k}\Big|_{\mathbf{w}=\mathbf{w}^*} \\ & = \frac{s(\mathbf{w})}{t(\mathbf{w})} \sum_{\mathbf{x}_- \in \mathcal{S}_-} \exp(\beta g(\mathbf{x}_-; \mathbf{w})) \frac{\partial g(\mathbf{x}_-; \mathbf{w})}{\partial w_k}\Big|_{\mathbf{w}=\mathbf{w}^*} \\ & = \sum_{\mathbf{x}_+ \in \mathcal{S}_+} \exp(-\alpha g(\mathbf{x}_+; \mathbf{w})) \frac{\partial g(\mathbf{x}_+; \mathbf{w})}{\partial w_k}\Big|_{\mathbf{w}=\mathbf{w}^*}, \ \forall k \in \{1, \dots, d\}. \end{aligned} \tag{B.9}$$

By Eq. (B.6) and (B.9), we have

$$\frac{\partial R^{\text{BC}}}{\partial w_k}\Big|_{\mathbf{w}=\mathbf{w}^*, b=b^*} = 0, \ \forall k \in \{1, \dots, d\}. \tag{B.10}$$

Lastly, by Eq. (B.8) and (B.10), we have $(\mathbf{w}^*, b^*) \in \operatorname{argmin}_{\mathbf{w},b} R^{\text{BC}}(f, \mathcal{D})$.

On the other hand, suppose $(\mathbf{w}^*, \tilde{b}) \in \operatorname{argmin}_{\mathbf{w},b} R^{\text{BC}}(f, \mathcal{D})$, by Eq. (B.4) we have

$$0 = \frac{\partial R^{\text{BC}}}{\partial b}\Big|_{\mathbf{w}=\mathbf{w}^*, b=\tilde{b}} = -\exp(-\alpha \tilde{b}) s(\mathbf{w}^*) + C \exp(\beta \tilde{b}) t(\mathbf{w}^*),$$

or equivalently

$$\tilde{b} = \frac{1}{\alpha + \beta} \ln \frac{s(\mathbf{w}^*)}{t(\mathbf{w}^*) C} = b^*. \tag{B.11}$$

In addition, we have $0 = \frac{\partial R^{\mathrm{BC}}}{\partial w_k}\Big|_{\mathbf{w}=\mathbf{w}^*,b=b^*}$, and by Eq. (B.6)

$$\begin{aligned}
&\sum_{\mathbf{x}_+\in\mathcal{S}_+} \exp(-\alpha g(\mathbf{x}_+;\mathbf{w}))\frac{\partial g(\mathbf{x}_+;\mathbf{w})}{\partial w_k}\Big|_{\mathbf{w}=\mathbf{w}^*} \\
&= C\exp((\alpha+\beta)b^*)\sum_{\mathbf{x}_-\in\mathcal{S}_-} \exp(\beta g(\mathbf{x}_-;\mathbf{w}))\frac{\partial g(\mathbf{x}_-;\mathbf{w})}{\partial w_k}\Big|_{\mathbf{w}=\mathbf{w}^*} \\
&= \frac{s(\mathbf{w})}{t(\mathbf{w})}\sum_{\mathbf{x}_-\in\mathcal{S}_-} \exp(\beta g(\mathbf{x}_-;\mathbf{w}))\frac{\partial g(\mathbf{x}_-;\mathbf{w})}{\partial w_k}\Big|_{\mathbf{w}=\mathbf{w}^*},\ \forall k\in\{1,\ldots,d\}.
\end{aligned}$$

Thus,

$$\begin{aligned}
&s(\mathbf{w})\sum_{\mathbf{x}_-\in\mathcal{S}_-} \exp(\beta g(\mathbf{x}_-;\mathbf{w}))\frac{\partial g(\mathbf{x}_-;\mathbf{w})}{\partial w_k}\Big|_{\mathbf{w}=\mathbf{w}^*} \\
&= t(\mathbf{w})\sum_{\mathbf{x}_+\in\mathcal{S}_+} \exp(-\alpha g(\mathbf{x}_+;\mathbf{w}))\frac{\partial g(\mathbf{x}_+;\mathbf{w})}{\partial w_k}\Big|_{\mathbf{w}=\mathbf{w}^*},\ \forall k\in\{1,\ldots,d\}.
\end{aligned} \tag{B.12}$$

By Eq. (B.6) and (B.12), we have

$$\frac{\partial R^{\mathrm{BR}}}{\partial w_k}\Big|_{\mathbf{w}=\mathbf{w}^*} = 0,\ \forall k\in\{1,\ldots,d\}. \tag{B.13}$$

Finally, by Eq. (B.13), $\mathbf{w}^* \in \operatorname*{argmin}_{\mathbf{w}} R^{\mathrm{BR}}(f,\mathcal{D})$.

□

Appendix C

Time Constraints for Travel Trajectory Recommendation

We have discussed how to adapt our proposed travel trajectory recommendation methods in Chapters 3 and 4 to incorporate time constraints such as the time budget for a desired trajectory. It turns out we can further extend these methods to incorporate time-of-the-day constraints on POIs (i.e., a POI shall be visited in its available time frame in a day). Here, we detail this approach formally.

Let M be the number of POIs in a city, and s_i, e_i be the earliest and latest available time of POI p_i, $i \in \{1, \ldots, M\}$ in a day, respectively, in other words, one can only visit p_i in the time frame $[s_i, e_i]$. Further, let c_i denote the time to be spent at POI p_i and t_{ij} be the time to travel from p_i to p_j, and assume $c_i \leq (e_i - s_i)$.

Suppose a user would like to start travelling at time T_0, and assuming a recommended trajectory will not span more than one day. Given a query $q = (p_s, p_e, T)$ where p_s is the start place and p_e the end location, and we require the total travelling time shall not exceed the time budget T. Below, we detail an approach to incorporate these constraints in an ILP formulation similar to the one presented in Chapter 3.

Let z_{mij}, $i, j \in \{1, \ldots, M\}$, $m \in \{2, \ldots, M\}$ be binary variables, and $z_{mij} = 1$ indicates that p_i is the $(m-1)$-th visited POI and p_j the m-th visited POI in a suggested trajectory. First, we shall avoid self-loops at any POI,

$$\sum_{m=2}^{M} \sum_{i=1}^{M} z_{mii} = 0. \tag{C.1}$$

For brevity, POIs are rearranged such that $p_1 = p_s$ and $p_M = p_e$. Therefore, we can restrict that p_1 is the first POI to be visited via

$$\begin{aligned} \sum_{m=2}^{M} \sum_{i=1}^{M} z_{mi1} = 0, \quad & \text{(No in-coming transitions)} \\ \sum_{j=1}^{M} z_{21j} = 1. \quad & \text{(One out-going transition when } m = 2) \end{aligned} \tag{C.2}$$

We further constrain that p_M is the last POI to be visited via

$$\begin{aligned} &\sum_{m=2}^{M}\sum_{i=1}^{M} z_{miM} = 1, \quad \text{(One in-coming transition only)} \\ &\sum_{m=2}^{M}\sum_{j=1}^{M} z_{mMj} = 0. \quad \text{(No out-going transitions)} \end{aligned} \tag{C.3}$$

In addition, each POI shall be visited at most once

$$\sum_{m=2}^{M}\sum_{i=1}^{M} z_{mik} = \sum_{m=2}^{M}\sum_{j=1}^{M} z_{mkj} \leq 1, \ \forall k \in \{2,\dots,M-1\}. \tag{C.4}$$

Note that the definition of variables z_{mij} implies that we shall visit at most one POI at step m

$$\sum_{i=1}^{M}\sum_{j=1}^{M} z_{mij} \leq 1, \ \forall m \in \{2,\dots,M\} \tag{C.5}$$

and further, the step $m-1$ and step m in a desired trajectory should be coherent, i.e.,

$$\sum_{i=1}^{M} z_{m-1,i,k} = \sum_{j=1}^{M} z_{mkj}, \ \forall m \in \{3,\dots,M\}, \ \forall k \in \{2,\dots,M-1\}. \tag{C.6}$$

Suppose $s_1 \geq T_0$, and assume that POI p_k is the m'-th visited POI in a desired trajectory, where $k \in \{2,\dots,M\}$ and $m' \in \{2,\dots,M\}$, we have $\sum_{i=1}^{M} z_{m'ik} = 1$. The time before visiting p_k can be represented as

$$\begin{aligned} T_k = T_0 + \frac{c_1}{2} + \sum_{m=2}^{m'}\sum_{i=1}^{M}\sum_{j=1}^{M} z_{mij}\left(t_{ij} + \frac{c_i}{2} + \frac{c_j}{2}\right) - \frac{c_k}{2}, \\ k \in \{2,\dots,M\}, \ m' \in \{2,\dots,M\}. \end{aligned} \tag{C.7}$$

To account for the time-of-the-day constraints on p_k, we have $s_k \leq T_k \leq e_k$. To create a linear constraint that incorporates $s_k \leq T_k$ when $\sum_{i=1}^{M} z_{m'ik} = 1$, we first rearrange $s_k \leq T_k$ into an inequality

$$\sum_{m=2}^{m'}\sum_{i=1}^{M}\sum_{j=1}^{M} z_{mij}\left(2t_{ij} + c_i + c_j\right) \geq 2(s_k - T_0) + c_k - c_1, \tag{C.8}$$

we can then multiply the RHS of (C.8) by $\sum_{i=1}^{M} z_{m'ik}$, which results in

$$\sum_{m=2}^{m'}\sum_{i=1}^{M}\sum_{j=1}^{M} z_{mij}\left(2t_{ij} + c_i + c_j\right) \geq [2(s_k - T_0) + c_k - c_1]\sum_{i=1}^{M} z_{m'ik}. \tag{C.9}$$

Note that $\sum_{i=1}^{M} z_{m'ik} \in \{0,1\}$ by constraint (C.5), and the linear constraint (C.9) is identical to inequality (C.8) when $\sum_{i=1}^{M} z_{m'ik} = 1$. If $\sum_{i=1}^{M} z_{m'ik} = 0$, inequality (C.9) degrades to a trivial constraint, since the LHS of (C.9) is non-negative. Similarly, we can rearrange $T_k \leq e_k$ into an inequality

$$\sum_{m=2}^{m'} \sum_{i=1}^{M} \sum_{j=1}^{M} z_{mij} \left(2t_{ij} + c_i + c_j\right) \leq 2(e_k - T_0) + c_k - c_1, \tag{C.10}$$

and observe that if we replace the RHS of (C.10) with the following linear expression

$$[2(e_k - T_0) + c_k - c_1] \sum_{i=1}^{M} z_{m'ik} + C \cdot \left(1 - \sum_{i=1}^{M} z_{m'ik}\right),$$

where constant $C \in \mathbb{R}$ is a big number. We thus have a linear constraint

$$\sum_{m=2}^{m'} \sum_{i=1}^{M} \sum_{j=1}^{M} z_{mij} \left(2t_{ij} + c_i + c_j\right) \leq [2(e_k - T_0) + c_k - c_1] \sum_{i=1}^{M} z_{m'ik} \; + \; C \cdot \left(1 - \sum_{i=1}^{M} z_{m'ik}\right). \tag{C.11}$$

Note that constraint (C.11) is identical to inequality (C.10) when $\sum_{i=1}^{M} z_{m'ik} = 1$, and if $\sum_{i=1}^{M} z_{m'ik} = 0$, the RHS of (C.11) becomes C. If C is big enough, (C.11) degrades to a trivial constraint.

We can summarise the above discussion by the following ILP formulation. It optimises the linear objective $f(\mathbf{z}, \mathbf{u})$ and also takes care of the time budget constraint (i.e., $T_M + c_M - T_0 \leq T$) as well as avoiding sub-tours (Miller et al., 1960).

$$\begin{aligned}
\max_{\mathbf{z},\mathbf{u}} \quad & f(\mathbf{z}, \mathbf{u}) \\
s.t. \quad & z_{mij} \in \{0,1\}, \; \forall i,j \in \{1,\ldots,M\}, \; \forall m \in \{2,\ldots,M\} \\
& \sum_{m=2}^{M} \sum_{i=1}^{M} z_{mii} = 0, \quad \sum_{i=1}^{M} \sum_{j=1}^{M} z_{mij} \leq 1, \; \forall m \in \{2,\ldots,M\} \\
& \sum_{m=2}^{M} \sum_{i=1}^{M} z_{mi1} = 0, \quad \sum_{j=1}^{M} z_{21j} = 1, \\
& \sum_{m=2}^{M} \sum_{i=1}^{M} z_{miM} = 1, \quad \sum_{m=2}^{M} \sum_{j=1}^{M} z_{mMj} = 0, \\
& \sum_{m=2}^{M} \sum_{i=1}^{M} z_{mik} = \sum_{m=2}^{M} \sum_{j=1}^{M} z_{mkj} \leq 1, \; \forall k \in \{2,\ldots,M-1\} \\
& \sum_{i=1}^{M} z_{m-1,i,k} = \sum_{j=1}^{M} z_{mkj}, \; \forall m \in \{3,\ldots,M\}, \; \forall k \in \{2,\ldots,M-1\}
\end{aligned}$$

$$\sum_{m=2}^{m'}\sum_{i=1}^{M}\sum_{j=1}^{M} z_{mij}\left(2t_{ij}+c_i+c_j\right) \geq [2(s_k-T_0)+c_k-c_1]\sum_{i=1}^{M} z_{m'ik},$$

$$\forall k \in \{2,\ldots,M\},\ \forall m' \in \{2,\ldots,M\}$$

$$\sum_{m=2}^{m'}\sum_{i=1}^{M}\sum_{j=1}^{M} z_{mij}\left(2t_{ij}+c_i+c_j\right) \leq [2(e_k-T_0)+c_k-c_1]\sum_{i=1}^{M} z_{m'ik} \ + \ C\left(1-\sum_{i=1}^{M} z_{m'ik}\right),$$

$$\forall k \in \{2,\ldots,M\},\ \forall m' \in \{2,\ldots,M\}$$

$$\sum_{m=2}^{M}\sum_{i=1}^{M}\sum_{j=1}^{M} z_{mij}\left(2t_{ij}+c_i+c_j\right) \leq 2T-c_M-c_1,$$

$$u_i \in \mathbb{Z},\ \forall i \in \{1,\ldots,M\}$$

$$u_i-u_j+1 \leq (M-1)\left(1-\sum_{m=2}^{M} z_{mij}\right),\ \forall i,j \in \{1,\ldots,M\}.$$

One example of the objective function is

$$f(\mathbf{z},\mathbf{u}) = \sum_{m=2}^{M}\sum_{i=1}^{M-1}\sum_{j=2}^{M} z_{mij}\log \mathbb{P}(p_j|p_i),$$

which is an extension of (3.5) that optimises the transition likelihood of the suggested trajectory.

Similarly, the above ILP formulation can be adapted to the setting in Chapter 4, where a query specifies a start location p_s and the time budget T. However, the end point of a desired trajectory needs particular care.

Lastly, we want to remark that the number of decision variables in the ILP (3.5)-(3.10) is on the order of $O(M^2)$ while that of the ILP formulation here is on the order of $O(M^3)$, and the latter could be significantly harder to optimise in general.

Appendix D

Details of Travel Trajectory Recommendation Experiments

D.1 Features

D.1.1 POI-query features

Table D.1 presents the features used to rank POIs with respect to a query. If the query includes the start and end points as well as the required number of POIs, i.e., (p_s, p_e, L), all features in Table D.1 are used to represent a POI-query pair; otherwise, if no end point is provided, i.e., a query is described as the start point p_s and trajectory length L, all features related to the end point in Table D.1 will be excluded.

The POI-query features are scaled to range $[-1.0, 1.0]$ using the approach that employed by libsvm,[1] i.e., fitting a linear function $f(x) = ax + b$ for feature x such that the maximum value of x maps to 1.0 and the minimum value maps to -1.0.

D.1.2 Transition features

We describe transitions among POIs using transition probabilities, in particular, we compute the POI-POI transition probabilities by first grouping POIs according to many POI features, as listed in Table D.2. For each feature in Table D.2, if the value of the feature is not categorical, it is discretised as described in Section 3.4, we then compute a transition matrix for the feature using maximum likelihood estimation, i.e., counting the number of POI transitions between the (discretised) feature values then normalising each row, while taking care of zeros by adding a small number ϵ (e.g., $\epsilon = 1$) to each count before normalisation.

In Chapter 3, the POI-POI transition matrix is computed by taking the Kronecker product of the transition matrices for the individual features, and then updating

[1] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

Table D.1: Features of POI p with respect to query $q = (p_s, p_e, L)$.

Feature	Description
`category`	one-hot encoding of the category of p
`neighbourhood`	one-hot encoding of the POI cluster that p resides in
`popularity`	logarithm of POI popularity of p
`nVisit`	logarithm of the total number of visits by all users at p
`avgDuration`	logarithm of the average visit duration at p
`trajLen`	trajectory length L, i.e., the number of POIs required
`sameCatStart`	1 if the category of p is the same as that of p_s, -1 otherwise
`sameNeighbourhoodStart`	1 if p resides in the same POI cluster as p_s, -1 otherwise
`distStart`	distance between p and p_s, calculated using the Haversine formula
`diffPopStart`	real-valued difference in POI popularity of p from that of p_s
`diffNVisitStart`	real-valued difference in the total number of visits at p from that at p_s
`diffDurationStart`	real-valued difference in average duration at p from that at p_s
`sameCatEnd`	1 if the category of p is the same as that of p_e, -1 otherwise
`sameNeighbourhoodEnd`	1 if p resides in the same POI cluster as p_e, -1 otherwise
`distEnd`	distance between p and p_e, calculated using the Haversine formula
`diffPopEnd`	real-valued difference in POI popularity of p from that of p_e
`diffNVisitEnd`	real-valued difference in the total number of visits at p from that at p_e
`diffDurationEnd`	real-valued difference in average visit duration at p from that at p_e

Table D.2: POI features used to capture POI-POI transitions.

Feature	Description
`category`	category of a POI
`neighbourhood`	the cluster that a POI resides in
`popularity`	(discretised) popularity of a POI
`nVisit`	(discretised) total number of visits at a POI
`avgDuration`	(discretised) average visit duration at a POI

it with three constraints (Section 3.4), which is based on our assumption that the transition probability from POI p_i to POI p_j can be factorised into the product of the transition probabilities between pairs of individual POI features, which are shown in Table D.2. In Chapter 4, the transition probabilities for each feature in Table D.2 are employed as pairwise features between consecutive POIs in a trajectory.

D.2 Evaluation metrics

Metrics used to evaluate the performance of trajectory recommendation methods including F_1 score on points (Lim et al., 2015), F_1 score on pairs (i.e., pairs-F_1) (Chen et al., 2016), and the well-known rank correlation Kendall's τ (Agresti, 2010).

D.2.1 F_1 score on points

F_1 score on points is the harmonic mean of precision and recall of POIs in trajectory,

$$F_1(\mathbf{y}, \hat{\mathbf{y}}) = \frac{2P_{\text{point}}R_{\text{point}}}{P_{\text{point}} + R_{\text{point}}},$$

where P_{POINT}, R_{POINT} are respectively the precision and recall for points in $\hat{\mathbf{y}}$ and $\mathbf{y}$,

$$P_{\text{point}} = \frac{|\mathcal{P}_{\mathbf{y}} \cap \mathcal{P}_{\hat{\mathbf{y}}}|}{|\mathcal{P}_{\hat{\mathbf{y}}}|},\ R_{\text{point}} = \frac{|\mathcal{P}_{\mathbf{y}} \cap \mathcal{P}_{\hat{\mathbf{y}}}|}{|\mathcal{P}_{\mathbf{y}}|}.$$

A perfect F_1 (i.e., $F_1 = 1$) means the set of POIs in the recommended trajectory are exactly the same as the set of POIs in the ground truth, and $F_1 = 0$ means that none of the POIs in the real trajectory was recommended.

While F_1 score on points is good at measuring whether POIs are correctly recommended, it ignores the visiting order between POIs. If $|\hat{\mathbf{y}}| = |\mathbf{y}|$, this metric is the same as the unordered Hamming loss, i.e., Hamming loss between two binary indicator vectors of size $|\mathcal{P}|$.

D.2.2 F_1 score on pairs

The F_1 score on pairs (i.e., pairs-F_1) takes into account both the point identity and the visiting orders in a trajectory. This is done by measuring the F_1 score of every pair of ordered POIs, whether they are adjacent or not in a trajectory,

$$\text{pairs-F}_1(\mathbf{y}, \hat{\mathbf{y}}) = \frac{2P_{\text{pair}}R_{\text{pair}}}{P_{\text{pair}} + R_{\text{pair}}},$$

where P_{pair} and R_{pair} are respectively the precision and recall for POI pairs in $\hat{\mathbf{y}}$ and $\mathbf{y}$,

$$P_{\text{pair}} = \frac{N_c}{|\mathcal{P}_{\hat{\mathbf{y}}}|(|\mathcal{P}_{\hat{\mathbf{y}}}| - 1)/2},\ R_{\text{pair}} = \frac{N_c}{|\mathcal{P}_{\mathbf{y}}|(|\mathcal{P}_{\mathbf{y}}| - 1)/2},$$

and N_c is the number of ordered POI pairs (p_i, p_j) that appear in both the ground-truth and the recommended trajectories, i.e.,

$$N_c = |\{(p_i, p_j) : p_i \prec_{\mathbf{y}} p_j,\ p_i, p_j \in \mathcal{P}_{\mathbf{y}}\} \cap \{(p_i, p_j) : p_i \prec_{\hat{\mathbf{y}}} p_j,\ p_i, p_j \in \mathcal{P}_{\hat{\mathbf{y}}}\}|,$$

where $p_i \prec_{\mathbf{y}} p_j$ denotes that POI p_i was visited before p_j in trajectory $\mathbf{y}$. We define pairs-$F_1 = 0$ when $N_c = 0$.

Pairs-F_1 takes values between 0 and 1. A perfect pairs-F_1 (1.0) is achieved if and only if both the POIs and their visiting orders in the recommended trajectory

are exactly the same as those in the ground truth. Pairs-$F_1 = 0$ means none of the recommended POI pairs was actually visited in the real trajectory.

D.2.3 Kendall's τ with ties

Alternatively, we can cast a trajectory $\mathbf{y} = y_{1:L}$ as a ranking of POIs in $\mathcal{P}$, where y_j has a rank $|\mathcal{P}| - j + 1$ and any other POI $p \notin \mathbf{y}$ has a rank 0 (0 is an arbitrary choice), then we can make use of ranking evaluation metrics such as Kendall's τ, while taking care of ties in ranks. In particular, given a prediction $\hat{\mathbf{y}} = \hat{y}_{1:L}$ and ground truth $\mathbf{y} = y_{1:L}$, we produce two ranks for $\mathbf{y}$ and $\hat{\mathbf{y}}$ with respect to a specific ordering of POIs $(p_1, \ldots, p_{|\mathcal{P}|})$:

$$r_i = \sum_{j=1}^{l} (|\mathcal{P}| - j + 1) [\![p_i = y_j]\!],\ i = 1, \ldots, |\mathcal{P}|$$
$$\hat{r}_i = \sum_{j=1}^{l} (|\mathcal{P}| - j + 1) [\![p_i = \hat{y}_j]\!],\ i = 1, \ldots, |\mathcal{P}|$$

where POIs not in $\mathbf{y}$ will have a rank of 0. Then we compute the following metrics:

- the number of concordant pairs $C = \frac{1}{2} \sum_{i,j} \left([\![r_i < r_j]\!][\![\hat{r}_i < \hat{r}_j]\!] + [\![r_i > r_j]\!][\![\hat{r}_i > \hat{r}_j]\!] \right)$
- the number of discordant pairs $D = \frac{1}{2} \sum_{i,j} \left([\![r_i < r_j]\!][\![\hat{r}_i > \hat{r}_j]\!] + [\![r_i > r_j]\!][\![\hat{r}_i < \hat{r}_j]\!] \right)$
- the number of ties in ground truth $\mathbf{y}$: $T_{\mathbf{y}} = \frac{1}{2} \sum_{i \neq j} [\![r_i = r_j]\!] = \frac{1}{2} \left(|\mathcal{P}| - L \right) \left(|\mathcal{P}| - L - 1 \right)$
- the number of ties in prediction $\hat{\mathbf{y}}$: $T_{\hat{\mathbf{y}}} = \frac{1}{2} \sum_{i \neq j} [\![\hat{r}_i = \hat{r}_j]\!] = \frac{1}{2} \left(|\mathcal{P}| - L \right) \left(|\mathcal{P}| - L - 1 \right)$
- the number of ties in both $\mathbf{y}$ and $\hat{\mathbf{y}}$: $T_{\mathbf{y},\hat{\mathbf{y}}} = \frac{1}{2} \sum_{i \neq j} [\![r_i = r_j]\!][\![\hat{r}_i = \hat{r}_j]\!]$

Kendall's τ (version b) (Agresti, 2010) is defined as

$$\tau_b(\mathbf{y}, \hat{\mathbf{y}}) = \frac{C - D}{\sqrt{(C + D + T)(C + D + U)}},$$

where $T = T_{\mathbf{y}} - T_{\mathbf{y},\hat{\mathbf{y}}}$ and $U = T_{\hat{\mathbf{y}}} - T_{\mathbf{y},\hat{\mathbf{y}}}$.

D.3 Additional empirical results

The performance of baselines and structured recommendation methods for top-k ($k = 1, 3, 5, 10$) recommendations are shown in Tables D.3, D.4, D.5 and D.6. Higher scores are better, The **best** method for each dataset (i.e., each row in table) is shown in bold, the *second best* is shown in italic.

Table D.3: Performance of trajectory recommendation on best of top-1.

	Random	Popularity	PoiRank	Markov	SP	SPpath	SR	SRpath
				F_1 score on points				
Osaka	.459 ± .027	.601 ± .031	**.678 ± .037**	.630 ± .034	.555 ± .034	.558 ± .036	.638 ± .039	*.645 ± .040*
Glasgow	.478 ± .027	.681 ± .032	**.764 ± .027**	.654 ± .027	.604 ± .026	.653 ± .031	.741 ± .028	*.743 ± .028*
Toronto	.461 ± .020	.671 ± .021	*.756 ± .021*	.676 ± .021	.594 ± .023	.623 ± .023	.753 ± .023	**.757 ± .022**
Edinburgh	.495 ± .016	**.730 ± .017**	*.707 ± .017*	.515 ± .020	.715 ± .017	N/A	*.718 ± .017*	N/A
Melbourne	.344 ± .009	**.512 ± .013**	.508 ± .013	.362 ± .012	.494 ± .013	N/A	*.510 ± .013*	N/A
				F_1 score on pairs				
Osaka	.104 ± .037	.281 ± .051	**.428 ± .059**	.331 ± .053	.243 ± .052	.254 ± .055	.375 ± .059	*.401 ± .060*
Glasgow	.154 ± .035	.426 ± .051	**.545 ± .046**	.368 ± .045	.289 ± .042	.389 ± .048	.506 ± .048	*.516 ± .048*
Toronto	.143 ± .025	.381 ± .034	.503 ± .036	.391 ± .034	.299 ± .033	.340 ± .035	*.530 ± .037*	**.533 ± .037**
Edinburgh	.177 ± .019	**.451 ± .026**	.426 ± .026	.250 ± .026	*.435 ± .027*	N/A	.428 ± .026	N/A
Melbourne	.034 ± .007	.192 ± .017	*.198 ± .017*	.065 ± .011	.176 ± .016	N/A	**.202 ± .018**	N/A
				Kendall's τ				
Osaka	.420 ± .030	.566 ± .034	**.644 ± .040**	.600 ± .036	.525 ± .037	.525 ± .039	.608 ± .042	*.613 ± .044*
Glasgow	.430 ± .031	.644 ± .036	**.733 ± .030**	.623 ± .030	.564 ± .029	.615 ± .034	.708 ± .031	*.712 ± .031*
Toronto	.394 ± .025	.626 ± .023	*.714 ± .024*	.629 ± .023	.543 ± .026	.572 ± .026	.714 ± .026	**.717 ± .026**
Edinburgh	.394 ± .020	**.663 ± .020**	.645 ± .021	.580 ± .019	.642 ± .021	N/A	*.648 ± .020*	N/A
Melbourne	.315 ± .011	*.489 ± .013*	**.495 ± .014**	.388 ± .011	.469 ± .014	N/A	.487 ± .014	N/A

Table D.4: Performance of trajectory recommendation on best of top-3.

	Random	Popularity	PoiRank	Markov	SP	SPpath	SR	SRpath
				F_1 score on points				
Osaka	.587 ± .034	.691 ± .035	**.750 ± .039**	*.740 ± .037*	.656 ± .040	.724 ± .037	.735 ± .038	.723 ± .039
Glasgow	.598 ± .028	.722 ± .033	.803 ± .027	.711 ± .029	.698 ± .030	.716 ± .029	*.825 ± .026*	**.829 ± .026**
Toronto	.577 ± .022	.704 ± .023	.776 ± .021	.748 ± .021	.674 ± .023	.693 ± .023	**.784 ± .022**	*.780 ± .021*
Edinburgh	.590 ± .017	.745 ± .017	.732 ± .017	.632 ± .021	*.754 ± .017*	N/A	**.778 ± .016**	N/A
Melbourne	.406 ± .011	.545 ± .014	.544 ± .015	.426 ± .015	**.577 ± .016**	N/A	*.568 ± .015*	N/A
				F_1 score on pairs				
Osaka	.288 ± .055	.448 ± .058	**.578 ± .060**	.538 ± .060	.425 ± .062	.511 ± .059	*.549 ± .060*	.520 ± .059
Glasgow	.300 ± .043	.524 ± .053	.625 ± .046	.465 ± .048	.464 ± .049	.481 ± .048	*.666 ± .045*	**.678 ± .045**
Toronto	.281 ± .032	.473 ± .036	.572 ± .035	.517 ± .035	.429 ± .037	.461 ± .037	**.592 ± .036**	*.583 ± .036*
Edinburgh	.301 ± .025	.502 ± .027	*.507 ± .027*	.402 ± .029	.504 ± .027	N/A	**.536 ± .026**	N/A
Melbourne	.090 ± .012	.259 ± .020	.270 ± .021	.159 ± .018	**.315 ± .022**	N/A	*.297 ± .021*	N/A
				Kendall's τ				
Osaka	.556 ± .037	.666 ± .039	**.726 ± .042**	*.718 ± .039*	.630 ± .044	.698 ± .040	.711 ± .042	.697 ± .042
Glasgow	.563 ± .031	.693 ± .036	.781 ± .030	.684 ± .032	.666 ± .033	.688 ± .032	*.803 ± .029*	**.808 ± .030**
Toronto	.521 ± .026	.670 ± .025	.746 ± .023	.712 ± .023	.629 ± .027	.650 ± .027	**.753 ± .025**	*.749 ± .024*
Edinburgh	.504 ± .021	.686 ± .020	.680 ± .021	.684 ± .019	*.690 ± .020*	N/A	**.721 ± .019**	N/A
Melbourne	.379 ± .012	.524 ± .015	.532 ± .015	.459 ± .014	**.555 ± .017**	N/A	*.546 ± .016*	N/A

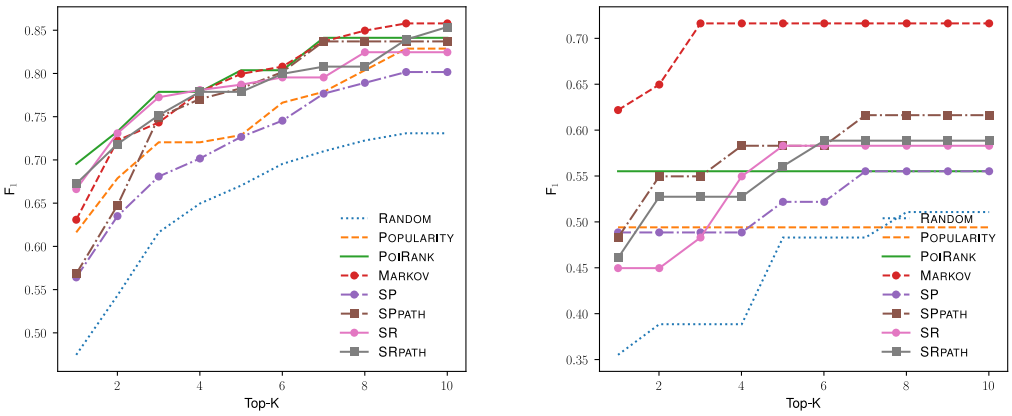
The performance of baselines and structured recommendation methods for short (length < 5) and long (length ≥ 5) trajectories with top-k ($k \in \{1, \ldots, 10\}$) recommendations on `Osaka` are shown in Figures D.1, D.2 and D.3; the performance on `Toronto` are shown in Figures D.4, D.5 and D.6; the performance on `Edinburgh` are shown in Figures D.7, D.8 and D.9; and the performance on `Melbourne` are shown in Figures D.10, D.11 and D.12.

Table D.5: Performance of trajectory recommendation on best of top-5.

	Random	Popularity	PoiRank	Markov	SP	SPpath	SR	SRpath
				F_1 score on points				
Osaka	.646 ± .035	.699 ± .034	*.772 ± .037*	**.789 ± .033**	.700 ± .041	*.757* ± .036	.761 ± .036	.751 ± .037
Glasgow	.655 ± .026	.754 ± .033	.821 ± .026	.736 ± .029	.755 ± .030	.770 ± .027	*.847 ± .024*	**.850 ± .025**
Toronto	.624 ± .022	.719 ± .023	.781 ± .021	.783 ± .021	.705 ± .023	.724 ± .022	**.808 ± .021**	*.798 ± .021*
Edinburgh	.646 ± .016	.747 ± .017	.739 ± .017	.655 ± .021	*.774 ± .017*	N/A	**.788 ± .016**	N/A
Melbourne	.450 ± .012	.556 ± .015	.570 ± .016	.451 ± .015	**.598 ± .016**	N/A	*.591 ± .016*	N/A
				F_1 score on pairs				
Osaka	.375 ± .058	.459 ± .057	*.607 ± .058*	**.621 ± .055**	.507 ± .064	.568 ± .058	.584 ± .058	.575 ± .058
Glasgow	.377 ± .044	.590 ± .052	.670 ± .045	.507 ± .048	.563 ± .048	.573 ± .047	*.701 ± .043*	**.715 ± .044**
Toronto	.343 ± .034	.500 ± .036	.590 ± .034	.581 ± .034	.483 ± .037	.509 ± .037	**.624 ± .035**	*.609 ± .035*
Edinburgh	.364 ± .025	.512 ± .027	.527 ± .027	.439 ± .029	*.540 ± .027*	N/A	**.561 ± .026**	N/A
Melbourne	.137 ± .015	.281 ± .021	.320 ± .022	.187 ± .019	**.348 ± .022**	N/A	*.337 ± .022*	N/A
				Kendall's τ				
Osaka	.618 ± .038	.674 ± .038	*.750 ± .040*	**.769 ± .036**	.678 ± .045	.735 ± .039	.741 ± .039	.729 ± .041
Glasgow	.623 ± .029	.727 ± .037	.801 ± .030	.712 ± .032	.727 ± .033	.743 ± .031	*.826 ± .028*	**.832 ± .028**
Toronto	.574 ± .025	.687 ± .025	.754 ± .023	.749 ± .024	.662 ± .027	.683 ± .026	**.778 ± .023**	*.769 ± .024*
Edinburgh	.562 ± .020	.693 ± .020	.692 ± .021	*.719 ± .018*	.714 ± .020	N/A	**.734 ± .019**	N/A
Melbourne	.425 ± .013	.535 ± .016	.560 ± .016	.485 ± .014	**.578 ± .017**	N/A	*.571 ± .017*	N/A

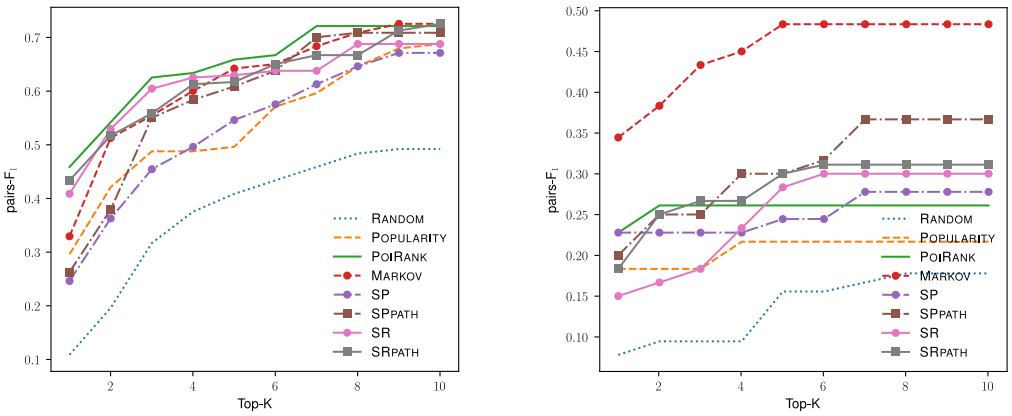
Table D.6: Performance of trajectory recommendation on best of top-10.

	Random	Popularity	PoiRank	Markov	SP	SPpath	SR	SRpath
				F_1 score on points				
Osaka	.703 ± .032	.786 ± .034	.804 ± .034	**.840 ± .029**	.770 ± .039	.809 ± .033	.793 ± .033	*.820 ± .031*
Glasgow	.731 ± .026	.771 ± .033	.847 ± .025	.800 ± .028	.810 ± .027	.807 ± .026	**.883 ± .023**	*.868 ± .023*
Toronto	.696 ± .021	.746 ± .022	.807 ± .020	.819 ± .019	.733 ± .023	.755 ± .022	**.828 ± .019**	*.823 ± .020*
Edinburgh	.689 ± .014	.768 ± .017	.754 ± .017	.679 ± .021	*.796 ± .016*	N/A	**.809 ± .015**	N/A
Melbourne	.511 ± .012	.587 ± .016	.609 ± .017	.508 ± .017	*.622 ± .016*	N/A	**.628 ± .016**	N/A
				F_1 score on pairs				
Osaka	.451 ± .057	.626 ± .055	.661 ± .056	**.693 ± .051**	.620 ± .061	.664 ± .055	.637 ± .055	*.671 ± .053*
Glasgow	.495 ± .046	.623 ± .051	.726 ± .043	.635 ± .048	.658 ± .046	.648 ± .045	**.770 ± .039**	*.746 ± .041*
Toronto	.438 ± .034	.546 ± .036	.646 ± .034	.644 ± .033	.530 ± .037	.552 ± .036	**.660 ± .033**	*.656 ± .034*
Edinburgh	.411 ± .024	.561 ± .027	.557 ± .027	.472 ± .030	*.588 ± .027*	N/A	**.608 ± .026**	N/A
Melbourne	.205 ± .018	.341 ± .023	.388 ± .023	.268 ± .023	*.390 ± .023*	N/A	**.403 ± .023**	N/A
				Kendall's τ				
Osaka	.685 ± .035	.768 ± .038	.787 ± .037	**.824 ± .031**	.749 ± .043	.791 ± .036	.777 ± .036	*.803 ± .034*
Glasgow	.703 ± .029	.748 ± .036	.830 ± .029	.781 ± .031	.790 ± .030	.787 ± .029	**.868 ± .026**	*.853 ± .026*
Toronto	.652 ± .024	.719 ± .024	.784 ± .023	.789 ± .022	.697 ± .027	.719 ± .026	**.802 ± .022**	*.797 ± .022*
Edinburgh	.616 ± .018	.718 ± .020	.712 ± .020	.739 ± .018	*.741 ± .020*	N/A	**.761 ± .018**	N/A
Melbourne	.488 ± .013	.567 ± .017	.601 ± .017	.541 ± .016	*.602 ± .017*	N/A	**.610 ± .017**	N/A



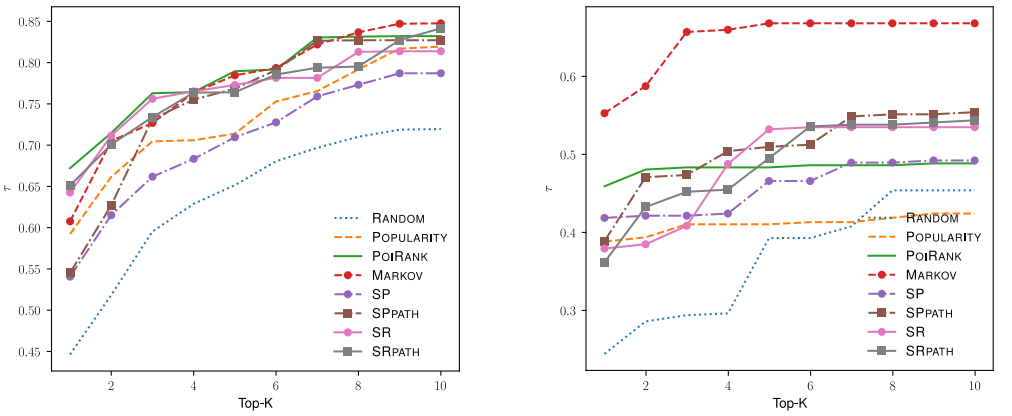
(a) Short Trajectories (b) Long Trajectories

Figure D.1: F_1 score on points over $k = 1:10$ for trajectories on Osaka.



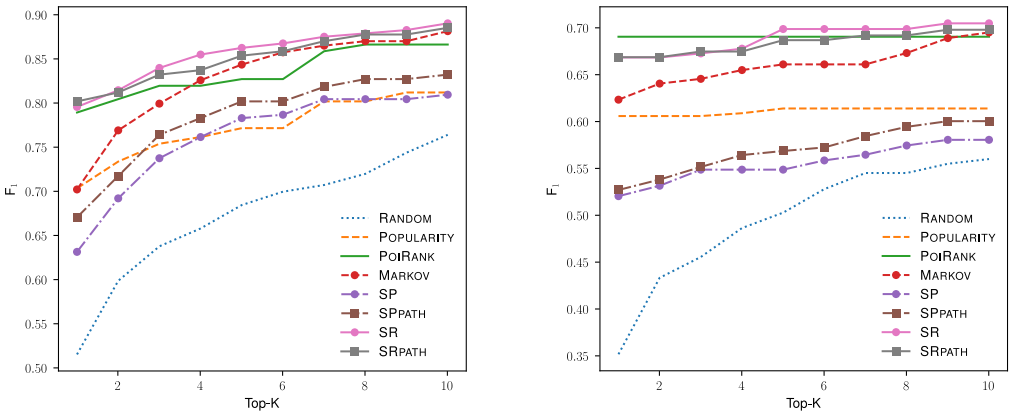
(a) Short Trajectories (b) Long Trajectories

Figure D.2: F_1 score on pairs over $k = 1:10$ for trajectories on Osaka.



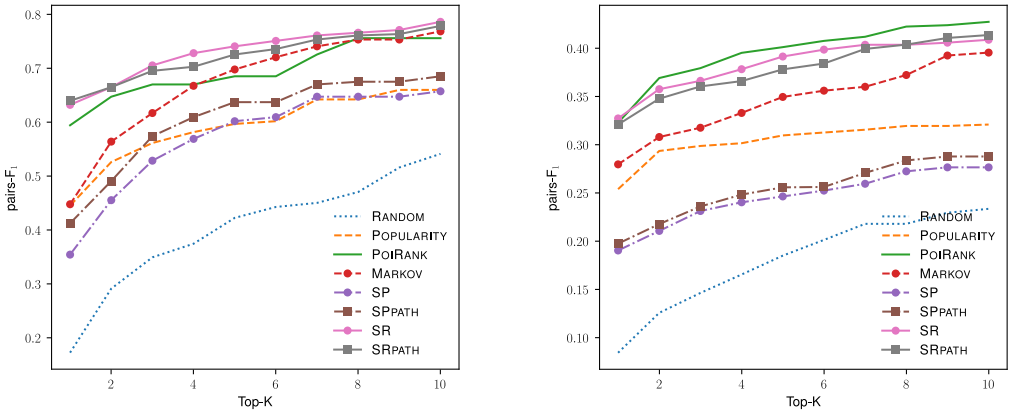
(a) Short Trajectories (b) Long Trajectories

Figure D.3: Kendall's τ over $k = 1:10$ for trajectories on Osaka.



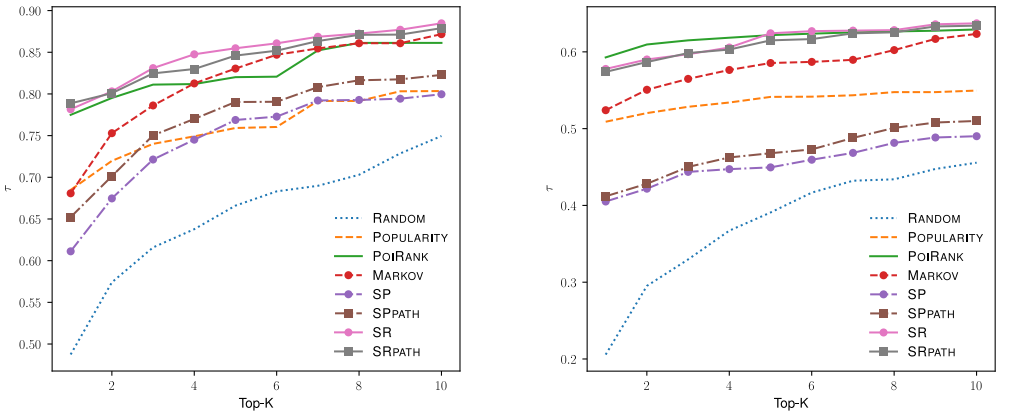
(a) Short Trajectories (b) Long Trajectories

Figure D.4: F_1 score on points over $k = 1:10$ for trajectories on Toronto.



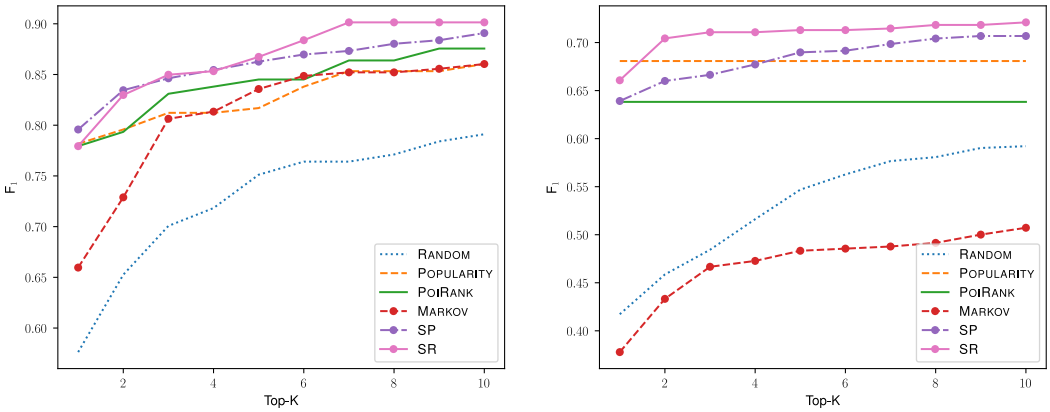
(a) Short Trajectories (b) Long Trajectories

Figure D.5: F_1 score on pairs over $k = 1:10$ for trajectories on Toronto.



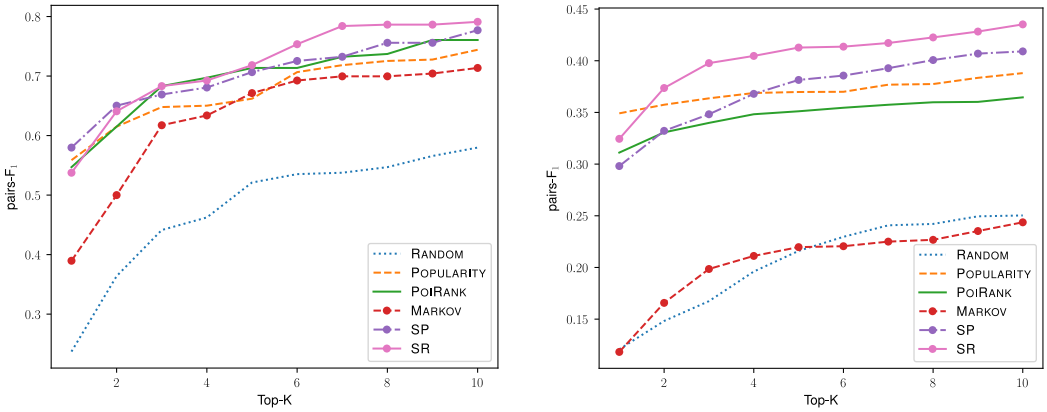
(a) Short Trajectories (b) Long Trajectories

Figure D.6: Kendall's τ over $k = 1:10$ for trajectories on Toronto.



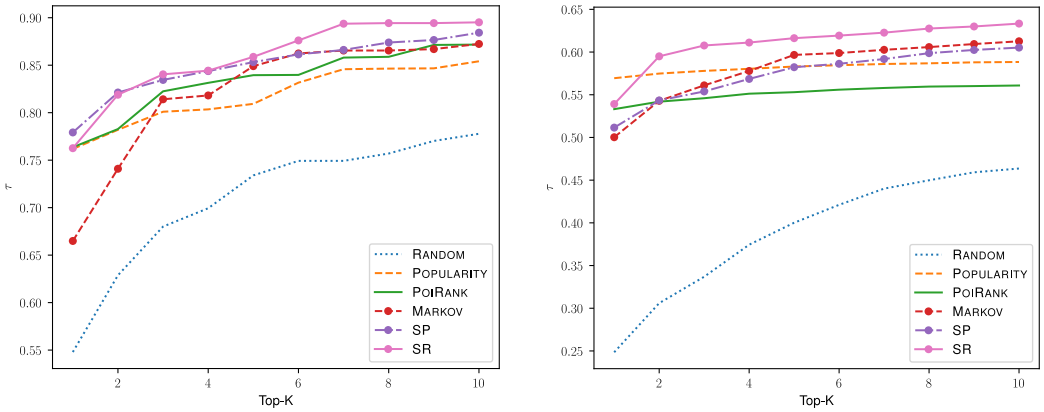
(a) Short Trajectories (b) Long Trajectories

Figure D.7: F_1 score on points over $k = 1:10$ for trajectories on Edinburgh.



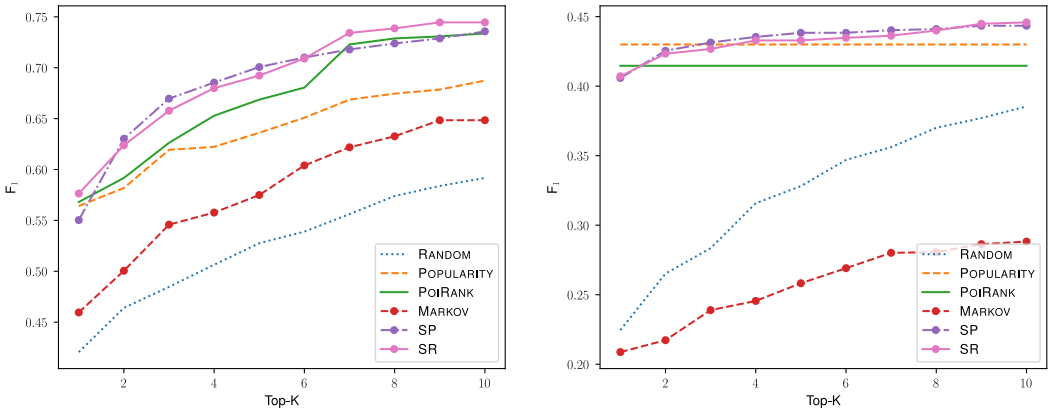
(a) Short Trajectories (b) Long Trajectories

Figure D.8: F_1 score on pairs over $k = 1:10$ for trajectories on Edinburgh.



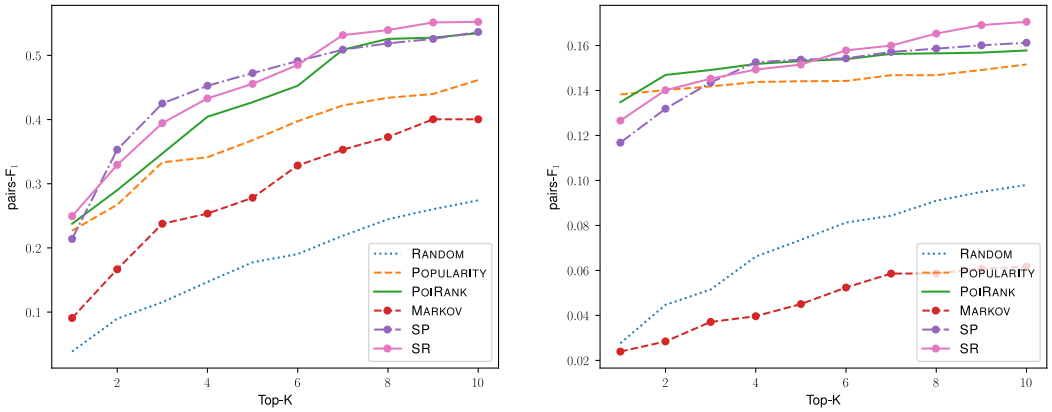
(a) Short Trajectories (b) Long Trajectories

Figure D.9: Kendall's τ over $k = 1:10$ for trajectories on Edinburgh.



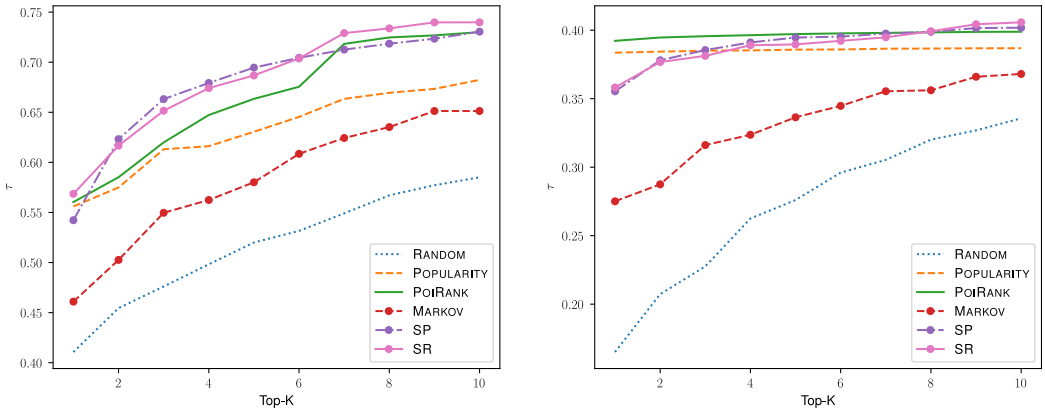
(a) Short Trajectories (b) Long Trajectories

Figure D.10: F_1 score on points over $k = 1:10$ for trajectories on Melbourne.



(a) Short Trajectories (b) Long Trajectories

Figure D.11: F_1 score on pairs over $k = 1:10$ for trajectories on Melbourne.



(a) Short Trajectories (b) Long Trajectories

Figure D.12: Kendall's τ over $k = 1:10$ for trajectories on Melbourne.

Proof of Lemma 1

First, we approximate the empirical risk R_θ^{RANK} (with the exponential surrogate loss) as follows:

$$
\begin{aligned}
R_\theta^{\text{RANK}}(f,\mathcal{D}) &= \frac{1}{N}\sum_{u=1}^{U}\sum_{i\in P_u}\frac{1}{M_-^i}\sum_{m':y_{m'}^i=0}\exp\left(-\min_{m:y_m^i=1} f(m,u,i)+f(m',u,i)\right)\\
&= \frac{1}{N}\sum_{u=1}^{U}\sum_{i\in P_u}\frac{1}{M_-^i}\exp\left(-\min_{m:y_m^i=1} f(m,u,i)\right)\sum_{m':y_{m'}^i=0}\exp\left(f(m',u,i)\right)\\
&\approx \frac{1}{N}\sum_{u=1}^{U}\sum_{i\in P_u}\frac{1}{M_-^i}\exp\left(\frac{1}{p}\log\sum_{m:y_m^i=1}e^{-pf(m,u,i)}\right)\sum_{m':y_{m'}^i=0}\exp\left(f(m',u,i)\right)\\
&= \frac{1}{N}\sum_{u=1}^{U}\sum_{i\in P_u}\frac{1}{M_-^i}\left(\sum_{m:y_m^i=1}e^{-pf(m,u,i)}\right)^{\frac{1}{p}}\sum_{m':y_{m'}^i=0}e^{f(m',u,i)}\\
&= \frac{1}{N}\sum_{u=1}^{U}\sum_{i\in P_u}\frac{1}{M_-^i}\left(\left(\sum_{m':y_{m'}^i=0}e^{f(m',u,i)}\right)^{p}\sum_{m:y_m^i=1}e^{-pf(m,u,i)}\right)^{\frac{1}{p}}\\
&= \frac{1}{N}\sum_{u=1}^{U}\sum_{i\in P_u}\frac{1}{M_-^i}\left(\sum_{m:y_m^i=1}e^{-pf(m,u,i)}\left(\sum_{m':y_{m'}^i=0}e^{f(m',u,i)}\right)^{p}\right)^{\frac{1}{p}}\\
&= \frac{1}{N}\sum_{u=1}^{U}\sum_{i\in P_u}\frac{1}{M_-^i}\left(\sum_{m:y_m^i=1}\left(\sum_{m':y_{m'}^i=0}e^{-(f(m,u,i)-f(m',u,i))}\right)^{p}\right)^{\frac{1}{p}}\\
&= \widetilde{R}_\theta^{\text{RANK}}(f,\mathcal{D}).
\end{aligned}
$$

Recall that R_θ^{MTC} is defined as

$$
R_\theta^{\text{MTC}}(f,\mathcal{D}) = \frac{1}{N}\sum_{u=1}^{U}\sum_{i\in P_u}\left(\frac{1}{pM_+^i}\sum_{m:y_m^i=1}e^{-pf(m,u,i)}+\frac{1}{M_-^i}\sum_{m':y_{m'}^i=0}e^{f(m',u,i)}\right).
$$

Let $\theta^* \in \operatorname{argmin}_\theta R_\theta^{\text{MTC}}$ (assuming minimisers exist), Lemma 1 in Section 5.4.4 shows that $\theta^* \in \operatorname{argmin}_\theta \widetilde{R}_\theta^{\text{RANK}}$.

Proof. We follow the proof technique in (Ertekin and Rudin, 2011) by first introducing a constant feature 1 for each song, without loss of generality, let the first feature of $\mathbf{x}_m$, $m \in \{1, \ldots, M\}$ be the constant feature, i.e., $x_m^0 = 1$. We then show that $\frac{\partial R_\theta^{\text{MTC}}}{\partial \theta} = 0$ implies $\frac{\partial \widetilde{R}_\theta^{\text{RANK}}}{\partial \theta} = 0$, which means minimisers of R_θ^{MTC} also minimise $\widetilde{R}_\theta^{\text{RANK}}$.

Let $0 = \dfrac{\partial R_\theta^{\text{MTC}}}{\partial \beta_i^0} = \dfrac{1}{N}\left(\dfrac{1}{pM_+^i}\displaystyle\sum_{m:y_m^i=1} e^{-pf(m,u,i)}(-p) + \dfrac{1}{M_-^i}\sum_{m':y_{m'}^i=0} e^{f(m',u,i)}\right)$, $i \in P_u$, $u \in \{1, \ldots, U\}$, we have

$$\frac{1}{M_+^i}\sum_{m:y_m^i=1} e^{-pf(m,u,i)}\bigg|_{\theta=\theta^*} = \frac{1}{M_-^i}\sum_{m':y_{m'}^i=0} e^{f(m',u,i)}\bigg|_{\theta=\theta^*}, \; i \in P_u, \; u \in \{1, \ldots, U\}. \tag{E.1}$$

Further, let

$$\mathbf{0} = \frac{\partial R_\theta^{\text{MTC}}}{\partial \boldsymbol{\beta}_i} = \frac{1}{N}\left(\frac{1}{pM_+^i}\sum_{m:y_m^i=1} e^{-pf(m,u,i)}(-p\mathbf{x}_m) + \frac{1}{M_-^i}\sum_{m':y_{m'}^i=0} e^{f(m',u,i)}\mathbf{x}_{m'}\right), \; i \in P_u, \; u \in \{1, \ldots, U\},$$

we have

$$\frac{1}{M_+^i}\sum_{m:y_m^i=1} e^{-pf(m,u,i)}\mathbf{x}_m\bigg|_{\theta=\theta^*} = \frac{1}{M_-^i}\sum_{m':y_{m'}^i=0} e^{f(m',u,i)}\mathbf{x}_{m'}\bigg|_{\theta=\theta^*}, \; i \in P_u, \; u \in \{1, \ldots, U\}. \tag{E.2}$$

By Eq. (E.1) and (E.2), for $i \in P_u$, $u \in \{1, \ldots, U\}$, we have

$$\begin{aligned}
&\frac{\partial \widetilde{R}_\theta^{\text{RANK}}}{\partial \boldsymbol{\beta}_i}\bigg|_{\theta=\theta^*} \\
&= \frac{1}{NM_-^i}\left[\frac{1}{p}\left(\sum_{m:y_m^i=1} e^{-pf(m,u,i)}\right)^{\frac{1}{p}-1}\sum_{m:y_m^i=1} e^{-pf(m,u,i)}(-p\mathbf{x}_m)\sum_{m':y_{m'}^i=0} e^{f(m',u,i)} + \left(\sum_{m:y_m^i=1} e^{-pf(m,u,i)}\right)^{\frac{1}{p}}\sum_{m':y_{m'}^i=0} e^{f(m',u,i)}\mathbf{x}_{m'}\right] \\
&= \frac{-1}{NM_-^i}\left(\sum_{m:y_m^i=1} e^{-pf(m,u,i)}\right)^{\frac{1}{p}-1}\left[\sum_{m:y_m^i=1} e^{-pf(m,u,i)}\mathbf{x}_m\sum_{m':y_{m'}^i=0} e^{f(m',u,i)} - \sum_{m:y_m^i=1} e^{-pf(m,u,i)}\sum_{m':y_{m'}^i=0} e^{f(m',u,i)}\mathbf{x}_{m'}\right] \\
&= \frac{-1}{NM_-^i}\left(\sum_{m:y_m^i=1} e^{-pf(m,u,i)}\right)^{\frac{1}{p}-1}\left[\left(\sum_{m:y_m^i=1} e^{-pf(m,u,i)}\mathbf{x}_m\right)\left(\frac{M_-^i}{M_+^i}\sum_{m:y_m^i=1} e^{-pf(m,u,i)}\right) - \sum_{m:y_m^i=1} e^{-pf(m,u,i)}\sum_{m':y_{m'}^i=0} e^{f(m',u,i)}\mathbf{x}_{m'}\right] \\
&= \frac{-1}{NM_-^i}\left(\sum_{m:y_m^i=1} e^{-pf(m,u,i)}\right)^{\frac{1}{p}}\left[\frac{M_-^i}{M_+^i}\sum_{m:y_m^i=1} e^{-pf(m,u,i)}\mathbf{x}_m - \sum_{m':y_{m'}^i=0} e^{f(m',u,i)}\mathbf{x}_{m'}\right] \\
&= \mathbf{0}.
\end{aligned} \tag{E.3}$$

We further let

$$h(u,i) = \frac{1}{NM_-^i}\left(\sum_{m:y_m^i=1} e^{-pf(m,u,i)}\right)^{\frac{1}{p}} \sum_{m':y_{m'}^i=0} e^{f(m',u,i)},\ i \in P_u,\ u \in \{1,\dots,U\},$$

and similar to Eq. (E.3), we have

$$\left.\frac{\partial h(u,i)}{\partial \boldsymbol{\beta}_i}\right|_{\theta=\theta^*} = \mathbf{0},\ i \in P_u,\ u \in \{1,\dots,U\}. \tag{E.4}$$

By Eq. (E.4), for $u \in \{1,\dots,U\}$, we have

$$\left.\frac{\partial \widetilde{R}_\theta^{\text{RANK}}}{\partial \boldsymbol{\alpha}_u}\right|_{\theta=\theta^*} = \sum_{i\in P_u}\left.\frac{\partial h(u,i)}{\partial \boldsymbol{\alpha}_u}\right|_{\theta=\theta^*} = \sum_{i\in P_u}\left.\frac{\partial h(u,i)}{\partial \boldsymbol{\beta}_i}\right|_{\theta=\theta^*} = \mathbf{0}, \tag{E.5}$$

and

$$\left.\frac{\partial \widetilde{R}_\theta^{\text{RANK}}}{\partial \boldsymbol{\mu}}\right|_{\theta=\theta^*} = \sum_{u=1}^{U}\sum_{i\in P_u}\left.\frac{\partial h(u,i)}{\partial \boldsymbol{\mu}}\right|_{\theta=\theta^*} = \sum_{u=1}^{U}\sum_{i\in P_u}\left.\frac{\partial h(u,i)}{\partial \boldsymbol{\beta}_i}\right|_{\theta=\theta^*} = \mathbf{0}. \tag{E.6}$$

Finally, by Eq. (E.3), (E.5), and (E.6), $\theta^* \in \operatorname{argmin}_\theta \widetilde{R}_\theta^{\text{RANK}}$. □

Bibliography

Abdel-Hamid, O., Mohamed, A.-r., Jiang, H., Deng, L., Penn, G., and Yu, D. (2014). Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22:1533–1545. (Cited on page 44.)

Agarwal, D. and Chen, B.-C. (2009). Regression-based latent factor models. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 19–28. (Cited on page 73.)

Agarwal, D., Chen, B.-C., Elango, P., and Ramakrishnan, R. (2013). Content recommendation on web portals. *Communications of the ACM*, 56:92–101. (Cited on page 51.)

Agarwal, S. (2011). The infinite push: A new support vector ranking algorithm that directly optimizes accuracy at the absolute top of the list. In *Proceedings of the IEEE International Conference on Data Mining*, pages 839–850. (Cited on pages 30 and 74.)

Agarwal, S. and Niyogi, P. (2005). Stability and generalization of bipartite ranking algorithms. In *Proceedings of the Annual Conference on Learning Theory*, pages 32–47. (Cited on pages 3 and 71.)

Aggarwal, C. C. (2016). *Recommender Systems: The Textbook*. Springer. (Cited on pages 1, 8, and 73.)

Aggarwal, C. C. and Parthasarathy, S. (2001). Mining massively incomplete data sets by conceptual reconstruction. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 227–232. (Cited on page 8.)

Agrafiotis, D. K., Bandyopadhyay, D., Wegner, J. K., and van Vlijmen, H. (2007). Recent Advances in Chemoinformatics. *Journal of chemical information and modeling*, 47:1279–1293. (Cited on pages 2 and 51.)

Agresti, A. (2010). *Analysis of ordinal categorical data*. Wiley. (Cited on pages 61, 112, and 114.)

Aji, S. M. and McEliece, R. J. (2000). The generalized distributive law. *IEEE transactions on Information Theory*, 46:325–343. (Cited on pages 17, 20, and 22.)

Amatriain, X. and Basilico, J. (2015). *Recommender Systems in Industry: A Netflix Case Study*, pages 385–419. Springer. (Cited on page 51.)

An, H.-C., Kleinberg, R., and Shmoys, D. B. (2015). Improving Christofides' Algorithm for the S-t Path TSP. *Journal of the ACM*, 62:34:1–34:28. (Cited on pages 3 and 26.)

Anagnostopoulos, A., Atassi, R., Becchetti, L., Fazzone, A., and Silvestri, F. (2017). Tour recommendation for groups. *Data Mining and Knowledge Discovery*, 31:1157–1188. (Cited on pages 36 and 66.)

Andrew, G. and Gao, J. (2007). Scalable training of L1-regularized log-linear models. In *Proceedings of the International Conference on Machine Learning*, pages 33–40. (Cited on page 79.)

Antikacioglu, A., Ravi, R., and Sridhar, S. (2015). Recommendation subgraphs for web discovery. In *Proceedings of the Web Conference*, pages 77–87. (Cited on page 51.)

Applegate, D. L., Bixby, R. E., Chvatal, V., and Cook, W. J. (2011). *The traveling salesman problem: a computational study*. Princeton University Press. (Cited on pages 2 and 3.)

Avriel, M. (2003). *Nonlinear programming: analysis and methods*. Courier Corporation. (Cited on page 78.)

Bach, S., Huang, B., Boyd-graber, J., and Getoor, L. (2015). Paired-dual learning for fast training of latent variable hinge-loss MRFs. In *Proceedings of the International Conference on Machine Learning*, pages 381–390. (Cited on page 16.)

BakIr, G., Hofmann, T., Schölkopf, B., Smola, A. J., Taskar, B., and Vishwanathan, S. (2007). *Predicting structured data*. MIT press. (Cited on pages 3 and 11.)

Bao, J., Zheng, Y., Wilkie, D., and Mokbel, M. (2015). Recommendations in location-based social networks: a survey. *GeoInformatica*, 19(3):525–565. (Cited on pages 34 and 35.)

Baraglia, R., Muntean, C. I., Nardini, F. M., and Silvestri, F. (2013). LearNext: learning to predict tourists movements. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 751–756. (Cited on page 36.)

Ben-Elazar, S., Lavee, G., Koenigstein, N., Barkan, O., Berezin, H., Paquet, U., and Zaccai, T. (2017). Groove radio: A bayesian hierarchical model for personalized playlist generation. In *Proceedings of the ACM International Conference on Web Search and Data Mining*, pages 445–453. (Cited on pages 2, 66, 73, 74, 75, 76, 83, and 92.)

Bennett, J., Lanning, S., et al. (2007). The Netflix Prize. In *Proceedings of KDD cup and workshop*. (Cited on page 1.)

Bertin-Mahieux, T., Ellis, D. P., Whitman, B., and Lamere, P. (2011). The million song dataset. In *Proceedings of the International Society for Music Information Retrieval*, pages 591–596. (Cited on page 79.)

Bonnin, G. and Jannach, D. (2013). Evaluating the quality of playlists based on hand-crafted samples. In *Proceedings of the International Society for Music Information Retrieval*, pages 263–268. (Cited on pages 73, 82, 83, 85, and 87.)

Bonnin, G. and Jannach, D. (2014). Automated generation of music playlists: Survey and experiments. *ACM Computing Surveys*, 47:26:1–26:35. (Cited on pages 66, 73, 83, 85, and 87.)

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press. (Cited on page 78.)

Boyd, S. and Vandenberghe, L. (2008). Localization and cutting-plane methods. http://web.stanford.edu/class/ee392o/localization-methods.pdf , retrieved September 2016. (Cited on pages 14, 97, 99, 101, and 102.)

Brilhante, I., Macedo, J. A., Nardini, F. M., Perego, R., and Renso, C. (2013). Where shall we go today?: planning touristic tours with tripbuilder. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 757–762. (Cited on pages 36 and 66.)

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. (2005). Learning to rank using gradient descent. In *Proceedings of the International Conference on Machine Learning*, pages 89–96. (Cited on page 2.)

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12:331–370. (Cited on page 73.)

Cao, B., Liu, N. N., and Yang, Q. (2010). Transfer learning for collective link prediction in multiple heterogenous domains. In *Proceedings of the International Conference on Machine Learning*, pages 159–166. (Cited on page 73.)

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., and Li, H. (2007). Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the International Conference on Machine Learning*, pages 129–136. (Cited on page 2.)

Caruana, R. (1993). Multitask learning: A knowledge-based source of inductive bias. In *Proceedings of the International Conference on Machine Learning*, pages 41–48. (Cited on pages 31 and 91.)

Caruana, R. (1997). Multitask learning. *Machine learning*, 28(1):41–75. (Cited on pages 31, 75, and 91.)

Charlesworth, B. and Charlesworth, D. (2010). *Elements of evolutionary genetics*. W. H. Freeman. (Cited on page 96.)

Chen, C., Zhang, D., Guo, B., Ma, X., Pan, G., and Wu, Z. (2015). TripPlanner: Personalized trip planning leveraging heterogeneous crowdsourced digital footprints. *IEEE Transactions on Intelligent Transportation Systems*, 16(3):1259–1273. (Cited on pages 36 and 37.)

Chen, D., Kim, D., Xie, L., Shin, M., Menon, A. K., Ong, C. S., Avazpour, I., and Grundy, J. (2017a). PathRec: Visual analysis of travel route recommendations. In *Proceedings of the ACM Recommender Systems Conference*, pages 364–365. (Cited on page 3.)

Chen, D., Ong, C. S., and Menon, A. K. (2019). Cold-start playlist recommendation with multitask learning. *CoRR*, abs/1901.06125. (Cited on page 3.)

Chen, D., Ong, C. S., and Xie, L. (2016). Learning points and routes to recommend trajectories. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 2227–2232. (Cited on pages 2, 3, 51, 59, 61, and 112.)

Chen, D., Xie, L., Menon, A. K., and Ong, C. S. (2017b). Structured recommendation. *CoRR*, abs/1706.09067. (Cited on pages 3 and 35.)

Chen, S., Moore, J. L., Turnbull, D., and Joachims, T. (2012). Playlist prediction via metric embedding. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 714–722. (Cited on pages 2, 51, 66, and 73.)

Chen, Y., Yan, W., Li, C., Huang, Y., and Yang, L. (2018). Personalized optimal bicycle trip planning based on q-learning algorithm. In *Wireless Communications and Networking Conference*, pages 1–6. (Cited on pages 36 and 66.)

Chen, Y.-Y., Cheng, A.-J., and Hsu, W. H. (2013). Travel recommendation by mining people attributes and travel group types from community-contributed photos. *IEEE Transactions on Multimedia*, 15(6):1283–1295. (Cited on page 37.)

Cheng, C., Yang, H., Lyu, M. R., and King, I. (2013). Where you like to go next: Successive point-of-interest recommendation. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 2605–2611. (Cited on pages 36 and 66.)

Choi, K., Fazekas, G., Sandler, M., et al. (2016). Towards playlist generation algorithms using RNNs trained on within-track transitions. *arXiv preprint arXiv:1606.02096*. (Cited on pages 2, 51, and 66.)

Christofides, N. (1976). Worst-case analysis of a new heuristic for the traveling salesman problem. *Sympos. on New Directions and Recent Results in Algorithms and Complexity*. (Cited on pages 27 and 58.)

Clémençon, S., Lugosi, G., Vayatis, N., et al. (2008). Ranking and empirical minimization of u-statistics. *The Annals of Statistics*, 36:844–874. (Cited on page 71.)

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press. (Cited on page 23.)

Cremonesi, P., Koren, Y., and Turrin, R. (2010). Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the ACM Recommender Systems Conference*, pages 39–46. (Cited on page 87.)

De Choudhury, M., Feldman, M., Amer-Yahia, S., Golbandi, N., Lempel, R., and Yu, C. (2010). Automatic construction of travel itineraries using social breadcrumbs. In *Proceedings of the ACM Conference on Hypertext and Social Media*, pages 35–44. (Cited on pages 38, 45, and 59.)

Debnath, M., Tripathi, P. K., Biswas, A. K., and Elmasri, R. (2018). Preference aware travel route recommendation with temporal influence. In *Proceedings of the ACM SIGSPATIAL Workshop on Recommendations for Location-based Services and Social Networks*. (Cited on page 36.)

Dehaspe, L., Toivonen, H., and King, R. D. (1998). Finding frequent substructures in chemical compounds. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 30–36. (Cited on pages 2 and 51.)

Dembczyński, K., Cheng, W., and Hüllermeier, E. (2010). Bayes optimal multilabel classification via probabilistic classifier chains. In *Proceedings of the International Conference on Machine Learning*, ICML'10, pages 279–286. (Cited on page 31.)

Deveaud, R., Albakour, M., Macdonald, C., Ounis, I., et al. (2014). On the importance of venue-dependent features for learning to rank contextual suggestions. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 1827–1830. (Cited on page 37.)

Deveaud, R., Albakour, M., Macdonald, C., Ounis, I., et al. (2015). Experiments with a venue-centric model for personalised and time-aware venue suggestion. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 53–62. (Cited on page 37.)

Diggle, P., Diggle, P. J., Heagerty, P., Heagerty, P. J., Liang, K.-Y., Zeger, S., et al. (2002). *Analysis of longitudinal data*. Oxford University Press. (Cited on page 75.)

Donaldson, J. (2007). A hybrid social-acoustic recommendation system for popular music. In *Proceedings of the ACM Recommender Systems Conference*, pages 187–190. (Cited on page 73.)

Dutka, J. (1991). The early history of the factorial function. *Archive for history of exact sciences*, 43(3):225–249. (Cited on page 25.)

Eghbal-Zadeh, H., Lehner, B., Schedl, M., and Widmer, G. (2015). I-vectors for timbre-based music similarity and music artist classification. In *Proceedings of the International Society for Music Information Retrieval*, pages 554–560. (Cited on page 73.)

Elzinga, J. and Moore, T. G. (1975). A central cutting plane algorithm for the convex programming problem. *Mathematical Programming*, 8(1):134–145. (Cited on page 100.)

Ertekin, Ş. and Rudin, C. (2011). On equivalence relationships between classification and ranking algorithms. *Journal of Machine Learning Research*, 12:2905–2929. (Cited on pages 3, 29, 30, 74, 78, 79, 103, and 122.)

Ference, G., Ye, M., and Lee, W.-C. (2013). Location recommendation for out-of-town users in location-based social networks. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 721–726. (Cited on page 37.)

Flickr (2019). The Flickr developer guide. https://www.flickr.com/services/developer , retrieved February 2019. (Cited on page 33.)

Foursquare (2019). FourSquare: about us. https://foursquare.com/about , retrieved February 2019. (Cited on page 33.)

Freund, Y., Iyer, R., Schapire, R. E., and Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969. (Cited on pages 30 and 71.)

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55:119–139. (Cited on page 29.)

Gantner, Z., Drumond, L., Freudenthaler, C., Rendle, S., and Schmidt-Thieme, L. (2010). Learning attribute-to-feature mappings for cold-start recommendations. In *Proceedings of the IEEE International Conference on Data Mining*, pages 176–185. (Cited on pages 74 and 83.)

Gao, H., Tang, J., Hu, X., and Liu, H. (2013). Exploring temporal effects for location recommendation on location-based social networks. In *Proceedings of the ACM Recommender Systems Conference*, pages 93–100. (Cited on pages 36 and 37.)

Gionis, A., Lappas, T., Pelechrinis, K., and Terzi, E. (2014). Customized tour recommendations in urban areas. In *Proceedings of the ACM International Conference on Web Search and Data Mining*, pages 313–322. (Cited on pages 35, 36, and 66.)

Goffin, J.-L. and Vial, J.-P. (2002). Convex nondifferentiable optimization: A survey focused on the analytic center cutting plane method. *Optimization Methods and Software*, 17:805–867. (Cited on page 101.)

Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. (1992). Using collaborative filtering to weave an information Tapestry. *Communications of the ACM*, 35:61–70. (Cited on pages 1 and 7.)

Golden, B. L., Levy, L., and Vohra, R. (1987). The Orienteering Problem. *Naval Research Logistics*, 34:307–318. (Cited on page 3.)

Gomez-Uribe, C. A. and Hunt, N. (2015). The Netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems*, 6:13:1–13:19. (Cited on page 51.)

Guo, S., Zoeter, O., and Archambeau, C. (2011). Sparse bayesian multi-task learning. In *Advances in Neural Information Processing Systems*, pages 1755–1763. (Cited on page 31.)

Hariri, N., Mobasher, B., and Burke, R. (2012). Context-aware music recommendation based on latenttopic sequential patterns. In *Proceedings of the ACM Recommender Systems Conference*, pages 131–138. (Cited on pages 66, 73, and 83.)

Hashem, T., Barua, S., Ali, M. E., Kulik, L., and Tanin, E. (2015). Efficient computation of trips with friends and families. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 931–940. (Cited on page 36.)

He, J., Qi, J., and Ramamohanarao, K. (2018). A jointly learned context-aware place of interest embedding for trip recommendations. *arXiv preprint arXiv:1808.08023*. (Cited on pages 2, 35, 36, 51, and 66.)

Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22:5–53. (Cited on page 83.)

Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. (2015). Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*. (Cited on pages 51 and 66.)

Hoogeveen, J. (1991). Analysis of Christofides' heuristic: Some paths are more difficult than cycles. *Operations Research Letters*, 10:291–295. (Cited on pages 3 and 26.)

Hsieh, H.-P. and Li, C.-T. (2014). Mining and planning time-aware routes from check-in data. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 481–490. (Cited on pages 36 and 37.)

Hu, B. and Ester, M. (2013). Spatial topic modeling in online social media for location recommendation. In *Proceedings of the ACM Recommender Systems Conference*, pages 25–32. (Cited on page 37.)

Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *Proceedings of the IEEE International Conference on Data Mining*, pages 263–272. (Cited on pages 9, 11, and 83.)

Huang, J.-T., Li, J., Yu, D., Deng, L., and Gong, Y. (2013). Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7304–7308. (Cited on page 31.)

Jannach, D., Lerche, L., and Kamehkhosh, I. (2015). Beyond hitting the hits: Generating coherent music playlist continuations with the right tracks. In *Proceedings of the ACM Recommender Systems Conference*, pages 187–194. (Cited on page 83.)

Joachims, T., Finley, T., and Yu, C.-N. J. (2009a). Cutting-plane training of structural SVMs. *Machine Learning*, 77:27–59. (Cited on pages 12, 13, and 14.)

Joachims, T., Hofmann, T., Yue, Y., and Yu, C.-N. (2009b). Predicting structured objects with support vector machines. *Communications of the ACM*, 52(11):97–104. (Cited on pages 2, 3, 14, 44, and 54.)

John, J. (2006). Pandora and the music genome project. *Scientific Computing*, 23:40–41. (Cited on pages 7 and 73.)

Jurafsky, D. and Martin, J. H. (2009). *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*. Prentice Hall. (Cited on pages 83 and 96.)

Kamehkhosh, I., Jannach, D., and Bonnin, G. (2018). How automated recommendations affect the playlist creation behavior of users. In *IUI Workshop on Intelligent Music Interfaces for Listening and Creation*. (Cited on page 92.)

Kluver, D. and Konstan, J. A. (2014). Evaluating recommender behavior for new users. In *Proceedings of the ACM Recommender Systems Conference*, pages 121–128. (Cited on page 84.)

Koren, Y. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 426–434. (Cited on page 11.)

Koren, Y. (2009). Collaborative filtering with temporal dynamics. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 447–456. (Cited on pages 1 and 11.)

Koren, Y. (2010). Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data*, 4:1:1–1:24. (Cited on page 51.)

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42:30–37. (Cited on pages 1, 2, 8, 9, 11, 66, and 69.)

Kotlowski, W., Dembczynski, K. J., and Huellermeier, E. (2011). Bipartite ranking through minimization of univariate loss. In *Proceedings of the International Conference on Machine Learning*, pages 1113–1120. (Cited on page 71.)

Kuo, T.-M., Lee, C.-P., and Lin, C.-J. (2014). Large-scale Kernel RankSVM. In *Proceedings of the SIAM International Conference on Data Mining*, pages 812–820. (Cited on page 3.)

Kurashima, T., Iwata, T., Irie, G., and Fujimura, K. (2010). Travel route recommendation using geotags in photo sharing sites. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 579–588. (Cited on page 37.)

Lacoste-julien, S., Jaggi, M., Schmidt, M., and Pletscher, P. (2013). Block-coordinate Frank-Wolfe optimization for structural SVMs. In *Proceedings of the International Conference on Machine Learning*, pages I–53–I–61. (Cited on page 17.)

Lafferty, J. D., McCallum, A., and Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*, pages 282–289. (Cited on page 44.)

Laird, N. M., Ware, J. H., et al. (1982). Random-effects models for longitudinal data. *Biometrics*, 38:963–974. (Cited on page 75.)

Le, Q. T. and Pishva, D. (2016). An innovative tour recommendation system for tourists in japan. In *International Conference on Advanced Communication Technology*, pages 717–729. (Cited on page 37.)

Lee, C.-P. and Lin, C.-b. (2014). Large-scale Linear RankSVM. *Neural computation*, 26(4):781–817. (Cited on pages 3 and 39.)

Li, N., Jin, R., and Zhou, Z.-H. (2014). Top rank optimization in linear time. In *Advances in Neural Information Processing Systems*, pages 1502–1510. (Cited on pages 30 and 74.)

Li, X., Pham, T.-A. N., Cong, G., Yuan, Q., Li, X.-L., and Krishnaswamy, S. (2015). Where you instagram?: Associating your instagram photos with points of interest. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 1231–1240. (Cited on page 36.)

Li, Y., Song, Y., and Luo, J. (2017). Improving pairwise ranking for multi-label image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. (Cited on page 31.)

Lian, D., Zhao, C., Xie, X., Sun, G., Chen, E., and Rui, Y. (2014). GeoMF: Joint geographical modeling and matrix factorization for point-of-interest recommendation. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 831–840. (Cited on pages 36 and 37.)

Lim, K. H., Chan, J., Leckie, C., and Karunasekera, S. (2015). Personalized tour recommendation based on user interests and points of interest visit durations. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1778–1784. (Cited on pages 2, 34, 35, 36, 44, 45, 47, 49, 51, 59, 61, 66, and 112.)

Linden, G., Smith, B., and York, J. (2003). Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing*, 7:76–80. (Cited on page 51.)

Liu, Q., Wu, S., Wang, D., Li, Z., and Wang, L. (2016a). Context-aware sequential recommendation. In *Proceedings of the IEEE International Conference on Data Mining*, pages 1053–1058. (Cited on page 44.)

Liu, Q., Wu, S., Wang, L., and Tan, T. (2016b). Predicting the next location: A recurrent model with spatial and temporal contexts. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 194–200. (Cited on pages 36, 44, and 66.)

Liu, T.-Y. (2009). Learning to Rank for Information Retrieval. *Foundations and Trends in Information Retrieval*, 3:225–331. (Cited on page 2.)

Liu, X., Liu, Y., Aberer, K., and Miao, C. (2013). Personalized point-of-interest recommendation by mining users' preference transition. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 733–738. (Cited on page 37.)

Liu, Y., Wei, W., Sun, A., and Miao, C. (2014). Exploiting geographical neighborhood characteristics for location recommendation. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 739–748. (Cited on pages 36 and 37.)

Louche, U. and Ralaivola, L. (2015). From cutting planes algorithms to compression schemes and active learning. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1–8. (Cited on page 101.)

Lu, E. H.-C., Chen, C.-Y., and Tseng, V. S. (2012). Personalized trip recommendation with multiple constraints by mining user check-in behaviors. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 209–218. (Cited on pages 2, 36, 37, and 51.)

Lu, X., Wang, C., Yang, J.-M., Pang, Y., and Zhang, L. (2010). Photo2Trip: Generating travel routes from geo-tagged photos for trip planning. In *Proceedings of the ACM Multimedia Conference*, pages 143–152. (Cited on pages 2, 36, and 51.)

Lu, Y.-S., Shih, W.-Y., Gau, H.-Y., Chung, K.-C., and Huang, J.-L. (2018). On successive point-of-interest recommendation. *World Wide Web*, pages 1–23. (Cited on page 36.)

Ma, H., Yang, H., Lyu, M. R., and King, I. (2008). SoRec: Social recommendation using probabilistic matrix factorization. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 931–940. (Cited on page 73.)

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press. (Cited on page 83.)

McFee, B., Bertin-Mahieux, T., Ellis, D. P., and Lanckriet, G. R. (2012). The Million Song Dataset Challenge. In *Proceedings of the Web Conference*, pages 909–916. (Cited on pages 73, 74, 82, and 87.)

McFee, B. and Lanckriet, G. R. (2011). The natural language of playlists. In *Proceedings of the International Society for Music Information Retrieval*, pages 537–542. (Cited on pages 2, 51, 66, and 73.)

McFee, B. and Lanckriet, G. R. (2012). Hypergraph models of playlist dialects. In *Proceedings of the International Society for Music Information Retrieval*, pages 343–348. (Cited on pages 73, 79, and 92.)

Menon, A. K. (2017). The Viterbi algorithm and its list variants. Technical report, Data61, CSIRO. (Cited on page 26.)

Menon, A. K., Chen, D., Xie, L., and Ong, C. S. (2017). Revisiting revisits in trajectory recommendation. In *Proceedings of the International Workshop on Recommender Systems for Citizens*, pages 2:1–2:6. (Cited on pages 3, 26, and 27.)

Menon, A. K. and Williamson, R. C. (2016). Bipartite ranking: a risk-theoretic perspective. *Journal of Machine Learning Research*, 17:1–102. (Cited on pages 3, 30, 74, and 78.)

Meshi, O., Sontag, D., Globerson, A., and Jaakkola, T. S. (2010). Learning efficiently with approximate inference via dual losses. In *Proceedings of the International Conference on Machine Learning*, pages 783–790. (Cited on page 16.)

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119. (Cited on page 80.)

Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM*, 7:326–329. (Cited on pages 2, 3, 27, 42, 56, and 109.)

Müller, A. (2014). *Methods for learning structured prediction in semantic segmentation of natural images*. PhD dissertation, Universitäts-und Landesbibliothek Bonn. (Cited on page 16.)

Narasimhan, H. and Agarwal, S. (2013). On the relationship between binary classification, bipartite ranking, and binary class probability estimation. In *Advances in Neural Information Processing Systems*, pages 2913–2921. (Cited on pages 74 and 78.)

Netflix (2006). Netflix Prize. http://www.netflixprize.com/. (Cited on page 69.)

Newman, M. (2010). *Networks: An Introduction*. Oxford University Press. (Cited on page 96.)

Nill, C. and Sundberg, C.-E. (1995). List and soft symbol output Viterbi algorithms: Extensions and comparisons. *IEEE Transactions on Communications*, 43:277–287. (Cited on pages 3 and 22.)

Nilsson, D. and Goldberger, J. (2001). Sequentially finding the N-best list in hidden Markov models. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1280–1285. (Cited on pages 3, 22, 23, 24, 26, 56, and 58.)

Oord, A. v. d., Dieleman, S., and Schrauwen, B. (2013). Deep content-based music recommendation. In *Advances in Neural Information Processing Systems*, pages 2643–2651. (Cited on pages 74 and 83.)

Ostendorf, M., Kannan, A., Austin, S., Kimball, O., Schwartz, R., and Rohlicek, J. R. (1991). Integration of diverse recognition methodologies through reevaluation of n-best sentence hypotheses. In *Proceedings of the Workshop on Speech and Natural Language*, pages 83–87. (Cited on page 22.)

Paterek, A. (2007). Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD cup and workshop*, pages 5–8. (Cited on pages 9 and 11.)

Platt, J. C., Burges, C. J., Swenson, S., Weare, C., and Zheng, A. (2001). Learning a gaussian process prior for automatically generating music playlists. In *Advances in Neural Information Processing Systems*, pages 1425–1432. (Cited on page 73.)

Pletscher, P., Ong, C. S., and Buhmann, J. M. (2010). Entropy and margin maximization for structured output learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 83–98. (Cited on page 11.)

Quercia, D., Schifanella, R., and Aiello, L. M. (2014). The shortest path to happiness: Recommending beautiful, quiet, and happy routes in the city. In *Proceedings of the ACM Conference on Hypertext and Social Media*, pages 116–125. (Cited on page 37.)

Ratliff, N., Bagnell, J. A., and Zinkevich, M. (2006). Subgradient methods for maximum margin structured learning. In *ICML workshop on learning in structured output spaces*. (Cited on page 17.)

Read, J., Pfahringer, B., Holmes, G., and Frank, E. (2009). Classifier chains for multi-label classification. In *Machine Learning and Knowledge Discovery in Databases*, pages 254–269. Springer Berlin Heidelberg. (Cited on page 31.)

Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. (2009). BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 452–461. (Cited on pages 11, 55, 66, and 74.)

Rendle, S., Freudenthaler, C., and Schmidt-Thieme, L. (2010). Factorizing personalized Markov chains for next-basket recommendation. In *Proceedings of the Web Conference*, pages 811–820. (Cited on pages 1, 36, and 66.)

Ruder, S. (2017). An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098. (Cited on pages 31 and 75.)

Rudin, C. (2009). The P-Norm Push: A simple convex ranking algorithm that concentrates at the top of the list. *Journal of Machine Learning Research*, 10:2233–2271. (Cited on pages 30, 74, 77, and 78.)

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115:211–252. (Cited on page 61.)

Salakhutdinov, R. and Mnih, A. (2008a). Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In *Proceedings of the International Conference on Machine Learning*, pages 880–887. (Cited on pages 10 and 11.)

Salakhutdinov, R. and Mnih, A. (2008b). Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems*, pages 1257–1264. (Cited on page 9.)

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the Web Conference*, pages 285–295. (Cited on page 69.)

Saveski, M. and Mantrach, A. (2014). Item cold-start recommendations: learning local collective embeddings. In *Proceedings of the ACM Recommender Systems Conference*, pages 89–96. (Cited on page 73.)

Schedl, M., Zamani, H., Chen, C.-W., Deldjoo, Y., and Elahi, M. (2017). Current challenges and visions in music recommender systems research. *ArXiv e-prints*. (Cited on pages 83, 84, and 92.)

Schein, A. I., Popescul, A., Ungar, L. H., and Pennock, D. M. (2002). Methods and metrics for cold-start recommendations. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 253–260. (Cited on page 2.)

Schindler, A., Mayer, R., and Rauber, A. (2012). Facilitating comprehensive benchmarking experiments on the million song dataset. In *Proceedings of the International Society for Music Information Retrieval*, pages 469–474. (Cited on page 80.)

Schreiber, H. (2015). Improving genre annotations for the million song dataset. In *Proceedings of the International Society for Music Information Retrieval*, pages 241–247. (Cited on page 80.)

Seshadri, N. and Sundberg, C.-E. (1994). List Viterbi decoding algorithms with applications. *IEEE Transactions on Communications*, 42:313–323. (Cited on pages 3, 22, 23, 24, 25, and 56.)

Seyerlehner, K., Widmer, G., Schedl, M., and Knees, P. (2010). Automatic music tag classification based on blocklevel features. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*. (Cited on page 73.)

Shao, B., Wang, D., Li, T., and Ogihara, M. (2009). Music recommendation based on acoustic features and user access patterns. *IEEE Transactions on Audio, Speech, and Language Processing*, 17:1602–1611. (Cited on page 73.)

Shi, Y., Serdyukov, P., Hanjalic, A., and Larson, M. (2011). Personalized landmark recommendation based on geotags from photo sharing sites. In *Proceedings of the International AAAI Conference on Web and Social Media*, pages 622–625. (Cited on page 36.)

Sinnott, R. W. (1984). Virtues of the haversine. *Sky and telescope*, 68(2):159. (Cited on pages 38 and 45.)

Soong, F. K. and Huang, E.-F. (1991). A tree-trellis based fast search for finding the n-best sentence hypotheses in continuous speech recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 705–708. (Cited on page 3.)

Spotify (2018). Company Info. https://newsroom.spotify.com/companyinfo , retrieved September 2018. (Cited on page 78.)

Stolcke, A., Konig, Y., and Weintraub, M. (1997). Explicit word error minimization in n-best list rescoring. In *European Conference on Speech Communication and Technology*. (Cited on page 22.)

Sutton, C., McCallum, A., et al. (2012). An introduction to conditional random fields. *Foundations and Trends® in Machine Learning*, 4:267–373. (Cited on page 44.)

Takács, G., Pilászy, I., Németh, B., and Tikk, D. (2007). Major components of the gravity recommendation system. *ACM SIGKDD Explorations Newsletter*, 9(2):80–83. (Cited on page 9.)

Taskar, B. (2004). *Learning structured prediction models: A large margin approach*. PhD dissertation, Stanford University. (Cited on pages 2 and 16.)

Taskar, B., Chatalbashev, V., Koller, D., and Guestrin, C. (2005). Learning structured prediction models: A large margin approach. In *Proceedings of the International Conference on Machine Learning*, pages 896–903. (Cited on pages 3 and 16.)

Taskar, B., Guestrin, C., and Koller, D. (2004). Max-margin markov networks. In *Advances in Neural Information Processing Systems*, pages 25–32. (Cited on page 44.)

Thomee, B., Elizalde, B., Shamma, D. A., Ni, K., Friedland, G., Poland, D., Borth, D., and Li, L.-J. (2016). YFCC100M: The new data in multimedia research. *Communications of the ACM*, 59(2):64–73. (Cited on pages 45 and 59.)

Tintarev, N., Lofi, C., and Liem, C. (2017). Sequences of diverse song recommendations: An exploratory study in a commercial system. In *Proceedings of the ACM Conference On User Modelling, Adaptation and Personalization*, pages 391–392. (Cited on page 92.)

Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the International Conference on Machine Learning*, pages 104–111. (Cited on pages 3, 14, and 44.)

Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484. (Cited on page 44.)

Turrin, R., Quadrana, M., Condorelli, A., Pagano, R., and Cremonesi, P. (2015). 30music listening and playlists dataset. In *Proceedings of the Poster Track of the ACM Conference on Recommender Systems*. (Cited on page 79.)

Vall, A., Schedl, M., Widmer, G., Quadrana, M., and Cremonesi, P. (2017). The importance of song context in music playlists. In *Proceedings of the Poster Track of the ACM Conference on Recommender Systems*. (Cited on page 92.)

Vapnik, V. (1992). Principles of risk minimization for learning theory. In *Advances in Neural Information Processing Systems*, pages 831–838. (Cited on page 75.)

Volkovs, M., Yu, G., and Poutanen, T. (2017). Dropoutnet: Addressing cold start in recommender systems. In *Advances in Neural Information Processing Systems*, pages 4957–4966. (Cited on page 73.)

Wallach, H. M. (2004). Conditional random fields: An introduction. *Technical Reports (CIS)*, page 22. (Cited on page 44.)

Wang, J., Yang, Y., Mao, J., Huang, Z., Huang, C., and Xu, W. (2016). CNN-RNN: A unified framework for multi-label image classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2285–2294. (Cited on page 31.)

Wang, P., Guo, J., Lan, Y., Xu, J., Wan, S., and Cheng, X. (2015). Learning hierarchical representation model for nextbasket recommendation. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 403–412. (Cited on page 66.)

Wu, X.-Z. and Zhou, Z.-H. (2017). A unified view of multi-label performance measures. In *Proceedings of the International Conference on Machine Learning*, pages 3780–3788. (Cited on page 31.)

Wulff, S. and Ong, C. S. (2013). Analytic center cutting plane method for multiple kernel learning. *Annals of Mathematics and Artificial Intelligence*, 69(3):225–241. (Cited on pages 14, 100, and 101.)

Xiong, L., Chen, X., HUang, T.-K., Schneider, J., and Carbonell, J. G. (2010). Temporal collaborative filtering with Bayesian probabilistic tensor factorization. In *Proceedings of the IEEE International Conference on Data Mining*, pages 211–222. (Cited on pages 1 and 11.)

Xue, Y., Dunson, D., and Carin, L. (2007). The matrix stick-breaking process for flexible multi-task learning. In *Proceedings of the International Conference on Machine Learning*, pages 1063–1070. ACM. (Cited on page 31.)

Yang, C., Sun, M., Zhao, W. X., Liu, Z., and Chang, E. Y. (2017). A neural network approach to jointly modeling social networks and mobile trajectories. *ACM Transactions on Information Systems (TOIS)*, 35:36. (Cited on page 44.)

Yoshii, K., Goto, M., Komatani, K., Ogata, T., and Okuno, H. G. (2006). Hybrid collaborative and content-based music recommendation using probabilistic model with latent user preferences. In *Proceedings of the International Society for Music Information Retrieval*. (Cited on page 73.)

Yu, F., Liu, Q., Wu, S., Wang, L., and Tan, T. (2016). A dynamic recurrent model for next basket recommendation. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 729–732. (Cited on page 66.)

Yuan, Q., Cong, G., Ma, Z., Sun, A., and Thalmann, N. M. (2013). Time-aware point-of-interest recommendation. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 363–372. (Cited on pages 36 and 37.)

Yuan, Q., Cong, G., and Sun, A. (2014). Graph-based point-of-interest recommendation with geographical and temporal influences. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 659–668. (Cited on pages 36 and 37.)

Zhang, C., Liang, H., Wang, K., and Sun, J. (2015a). Personalized trip recommendation with POI availability and uncertain traveling time. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 911–920. (Cited on page 37.)

Zhang, J.-D., Chow, C.-Y., and Li, Y. (2014). LORE: Exploiting sequential influence for location recommendations. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 103–112. (Cited on page 37.)

Zhang, J.-D., Chow, C.-Y., and Zheng, Y. (2015b). ORec: An opinion-based point-of-interest recommendation framework. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 1641–1650. (Cited on page 37.)

Zhang, W. and Wang, J. (2015). Location and time aware social collaborative retrieval for new successive point-of-interest recommendation. In *Proceedings of the ACM Conference on Information and Knowledge Management*, pages 1221–1230. (Cited on pages 36, 37, and 66.)

Zhang, Y. C., Séaghdha, D. Ó., Quercia, D., and Jambor, T. (2012). Auralist: introducing serendipity into music recommendation. In *Proceedings of the ACM International Conference on Web Search and Data Mining*, pages 13–22. (Cited on page 83.)

Zheng, Y. (2015). Trajectory data mining: An overview. *ACM Transactions on Intelligent Systems and Technology*, 6:29:1–29:41. (Cited on pages 34 and 35.)

Zheng, Y., Capra, L., Wolfson, O., and Yang, H. (2014). Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology*, 5:38:1–38:55. (Cited on pages 34 and 35.)

Zheng, Y., Zhang, L., Xie, X., and Ma, W.-Y. (2009). Mining interesting locations and travel sequences from GPS trajectories. In *Proceedings of the Web Conference*, pages 791–800. (Cited on page 36.)

Zheng, Y.-T., Zha, Z.-J., and Chua, T.-S. (2012). Mining travel patterns from geotagged photos. *ACM Transactions on Intelligent Systems and Technology*, 3(3):56:1–56:18. (Cited on page 36.)