FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Harnessing Spatial Context for Item Recommendation
Student:	Bc. Vendula Švastalová
Supervisor:	Rodrigo Augusto da Silva Alves, Ph.D.
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2024/2025

Instructions

Recommender systems (RSs) are designed to recommend items (e.g., movies, books, restaurants) to users. Some recommender systems operate in a spatial dimension. For example, a user residing in neighborhood A might rate a restaurant located in neighborhood B. Previous studies have explored spatial information in various ways, such as recommending based on points of interest, clustering recommender interactions by location, and incorporating spatial side information into the learning process. This project aims to study how spatial information in recommender systems can be used to propose a methods for recommending new items based on this spatial context. The tasks required for the completion of this project are as follows:

1) Conduct a comprehensive literature review to gain insights into related works.
2) Pre-process publicly available datasets and extract data from at least four cities for recommender systems with spatial side information.
3) Propose a method for recommending new items based on spatial information.
4) Evaluate the proposed method, demonstrating the quality of the results, and compare it with potential baseline methods.
5) Showcase interpretability aspects of the proposed method.

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 9 January 2024 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master’s thesis

Harnessing Spatial Context for Item Recommendation

Vendula Švastalová

Department of Applied Mathematics
Supervisor: Rodrigo Augusto da Silva Alves, Ph.D.

May 9, 2024

Acknowledgements

I would like to express my deepest gratitude to my thesis supervisor, Rodrigo Augusto da Silva Alves, Ph.D., for his invaluable patience and guidance throughout the duration of this thesis. His insights, expertise, and feedback were instrumental in making this thesis possible and have greatly helped me improve my skills and knowledge. I would also like to thank my partner and my family for their unwavering support and encouragement throughout my studies, which motivated me and made this journey profoundly rewarding.

Declaration

In Prague on May 9, 2024 . . . .

Citation of this thesis

Švastalová, Vendula. *Harnessing Spatial Context for Item Recommendation.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Abstrakt

Tato práce se zabývá vývojem doporučovacího systému který využívá polohová data uživatelů. Námi navržená architekrura kombinuje přístup kódování polohy se skupinami uživatelů a jejich preferencemi. Naše architektura se odlišuje tím, že zeměpisnou šířku a změpisnou délku kóduje do vektorového prostoru vyšší dimenze. Tento přístup umožňuje systému dynamicky generovat doporučení kategorií pro nové zájmové body, jako jsou události, místa, nebo aktivity v konkrétních lokalitách. Provedli jsme řadu experimentů, které ukazují, že navrhovaná architektura dosahuje zlepšení oproti základním modelům. Naše modely mohou být využity v sociálních sítích, kde mohou zvýšit zapojení jednotlivých uživatelů i komunity.

Klíčová slova Doporučovací systémy, GeoAI, Kódování polohy, Polohové doporučovací systémy, Strojové učení, Doporučování skupinám, Doporučování bodů zájmu

Abstract

This thesis explores the development of a location-based recommender system. We propose a novel architecture that integrates a general-purpose location encoder with groups of users and their preferences. Our architecture uses separate embeddings for latitude and longitude to dynamically generate recommendations of categories for new Points-of-Interests, such as events, venues, and activities at specific locations. We conduct a series of experiments to demonstrate that the proposed architecture outperforms baseline models. Our models offer more relevant and engaging content that can be used in location-based social networks, where it can increase user engagement and community involvement.

Keywords Recommender Systems, GeoAI, Location Encoding, Location-Based Services, Spatial Recommender Systems, Machine Learning, Group Recommendation, Point of Interest Recommendation

Contents

List of Figures

Introduction

"I invoke the first law of geography: everything is related to everything else, but near things are more related than distant things."

— Waldo Rudolph Tobler [1]

Motivation and Objectives

In today's digital age, the combination of location-based services with social networks has changed the way people connect with places around them. Despite the advancements, there remains room for these systems to better match dynamic user locations and their evolving preferences.

This thesis proposes the development of a location-based recommender system designed to automatically suggest the best features for creating new items such as local events, venues, or activities at specific location. For instance, consider the following:

- There is a vacant commercial space (ideal for establishing a new dining establishment) available in a calm neighborhood of the city. An entrepreneur is considering opening a restaurant in this location. Would an Italian or Chinese cuisine restaurant better suit the preferences of the local residents in this neighborhood?
- A municipality is interested in offering sports activities tailored to the preferences of the residents in a specific area. Considering the interests of the local citizens, would they prefer a running program or a cycling program?
- A local bookstore is organizing an itinerant book club to encourage reading and literary discussions among residents in different parts of the

city. Would the community in the next neighborhood prefer to focus on classic literature or contemporary bestsellers for their upcoming book club meeting?

These examples are recommendations on linking geographical coordinates to item preferences and they are the core of our model architecture. These recommendations aim to capture user interest and encourage participation, which is especially valuable in location-based social networks (LBSNs) where users form groups around shared interests.

Note that, in the context of current LBSNs, users are could be also the ones who typically create content in which they organize events, such as sporting activities, for others to join. This research sees an opportunity to boost user engagement by transitioning from user-generated to system-generated content. Such a system would streamline the process of organizing events by automatically generating appealing and relevant activities based on an analysis of users' locations and interests. For example, instead of users manually organizing events, the system could automatically suggest activities such as watching a soccer match at a local bar or participating in a nearby group hike, customized specifically to the preferences of users in the common area. By automatically creating content, the system makes the community more active and engaged, and increases participation by offering activities that users are more likely to enjoy and join.

Moreover, this system could enhance the field of recommender systems by integrating novel methods of location encoding with user preferences, focusing on dynamic, group-based scenarios. The expected result is a framework that not only improves user satisfaction through personalized suggestions but also stimulates increased community involvement. By addressing these needs, the proposed methods aims to contribute to the development of recommendation engines that enhance the connection between social networks and the physical world.

In conclusion, this work offers the following contributions:

1. We propose a general-purpose architecture of a location encoder using a novel approach to encoding location coordinates as two separate embeddings for latitude and longitude exclusively. We propose this novel approach in Section 3.2.3.

2. We propose a machine learning model for creating new items based on recommended categories to groups of users at a common location. The aim of this approach is to dynamically adapt to both geographic data and preferences of the user group. We propose this model in Section 3.2.

3. We conduct experiments on four real-world geographic datasets for two different tasks:

Category Probability Modeling: Recommending top categories for a potential new POI given a geographic location, based on the modeled probability of the categories.

Rating Probability Distribution Modeling: Recommending top categories for a potential new POI given a geographic location, based on the estimated probability distribution of rating values for categories at that location.

We formally define the tasks in in Section 3.1 and the experimets are described in Chapter 4

4. We test the model on unseen data and compare the proposed model with several baselines to demonstrate its effectiveness. These comparisons are show in Section 4.3.2

Thesis Organization

This thesis is organized as follows: Chapter 1 provides an introduction to the current state of recommender systems, group recommendation, point-of-interest recommendation, and location encoding methods. Chapter 2 offers insights into the raw datasets used in this study, detailing their processing and the reasoning behind each step taken, as well as an overview and statistics of the data. In Chapter 3, we define the problems to be solved and propose a general-purpose architecture for a location encoder model to address these problems. Chapter 4 describes the setup for our experiments, including the baseline methods, evaluation metrics, and the statement of the optimization problem. Finally, we evaluate the proposed model against the baselines, present the results, and analyze their implications.

Chapter 1

Theoretical Background

In this chapter, we first focus on general Recommender Systems, outlining their key principles, methodologies, and the challenges they address. We then dive into Group Recommender Systems, which help with decision-making in collective scenarios, and Point of Interest Recommender Systems, which use geographical data to suggest locations like restaurants, tourist attractions, and other venues. Finally, the chapter transitions into a discussion on the use of Location Embeddings in GeoAI. These embeddings represent geographical locations in a vector space, capturing the nuanced spatial relationships and characteristics that traditional models might overlook.

1.1 Recommender Systems

Recommender systems are a fundamental component of the digital world today. Their primary function is to predict and recommend items to users based on their preferences and historical interactions. These systems significantly improve user experiences by personalizing content in various contexts, including e-Commerce websites, video streaming services, and social media networks. The core aim is to present content that users are most likely to be interested in, interact with, or buy.

Central to recommender systems are the concepts of *users*, *items*, and the *interactions* between them. These interactions are either explicitly given by the user (e.g., rating in one-to five stars scale) or implicitly inferred from their behavior (e.g., a user viewed a product in an e-commerce website). A common challenge within recommender systems is the issue of data sparsity, where the system usually contains a wide range of items compared to the relatively few items each user interacts with. This imbalance can complicate the process of providing accurate and relevant recommendations.

The goal of a recommender system is to suggest items that maximize the likelihood of a user interacting with them. These interactions might include viewing a video, clicking on an advertisement, or purchasing an item.

1.1.1 Interactions

Recommender systems typically involve two classes of entities: *users* and *items*. Each user provides a rating or preference for items, which could be products, for example. Both users and items may have attributes, and these attributes can be used to find similarities between them.

In recommender systems, we distinguish between two types of interaction between users and items: explicit and implicit.

Explicit interactions are those where users are explicitly asked to provide their feedback. Examples include:

- **star ratings** - user can rate an item (e.g., a movie) with a value from a range, (eg. 1-5) when prompted,
- **like / dislike** - user leaves a positive or negative feedback when prompted.

Implicit interactions are those the system infers or aggregates based on user behavior within the system. Examples include:

- **click** - a user clicks on an item, such as a dress in a marketplace, indicating interest by viewing the detail page,
- **add to cart** - user adds an item to their cart, which suggests interest in the item, even if it is not purchased immediately,
- **play video / song** - user plays a video or song, stopping a video or song early may indicate disinterest,
- **buy an item** - user buys an item, directly indicating a strong interest,
- **like** - when there is no option to dislike, a 'like' is considered an implicit feedback indicating preference.

The interactions between users and items are typically stored in a structure known as the *utility matrix* (also called the user-item interaction matrix). This matrix is usually sparse, meaning that there are many items within the system that a user has never interacted with, leaving most of the matrix entries empty. The matrix is sparse because there is usually a vast array of items available in the system, and it is not possible for users to interact with most of them.

Figure 1.1 shows the differences between explicit and implicit feedback within the utility matrix. The rows of the matrix represent users, while the columns correspond to items in the system. In practical scenarios, this matrix is even more sparse, as users only interact with a small percentage of the available items. The empty fields in the matrix indicate items that have not

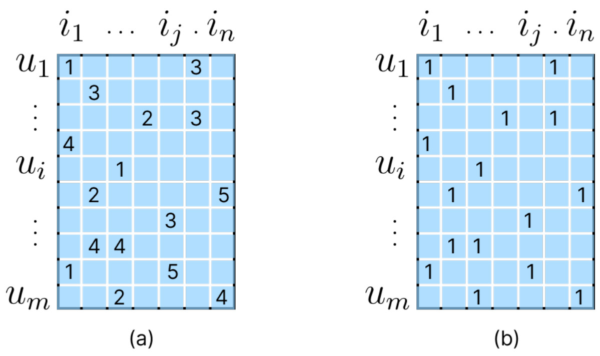

Figure 1.1: Utility matrices representing explicit (a) and implicit (b) feedback with rows representing the m users and columns representing the n items in the system.

yet been interacted with. These missing values pose a significant challenge for the system, which must predict them based on the available data.

The primary goal of the recommender system is to find the user-item pairs with the highest estimated interaction (rating) values. Generally the recommendation problem has been defined as an optimization problem [2]:

$$i^*(u) = \arg\max_{i \in \mathcal{I}} g(u, i) \tag{1.1}$$

where i^* is the item from the set of all items $\mathcal{I}$ that maximizes the relevance for user u and g represents the relevance function.

It is not necessary to determine all the unknown values. Because there are many item the system should only select a subset of items—those with the highest estimated score—to show to the users. This targeted approach ensures that users are presented only with the items most likely to interest them, rather than all items, focusing on the quality and relevance of recommendations.

1.1.2 Content-Based Recommendation

A content-based recommender system focuses on the features and properties of items within the system. It constructs item profiles based on these characteristics. Similarly, user profiles are created based on the historical interactions of users with items, reflecting the user's preferences.

The system uses these profiles to identify items that share similarities with those that a user has previously interacted with or shown interest in. Recommendations are then made to users based on these similar items. The users are then likely to be interested in the recommended items because they are similar to items which the user interacted with or liked.

Item profile is a collection of characteristics of an item that are relevant to the system. For example, in a video streaming service, items like movies might have features such as *genre*, *director*, or *length.* With advancements in natural language processing, some features like the cast or the year of the movie can be extracted directly from textual descriptions or reviews. Others, such as genre, might be assigned from predefined options within the system. For items that are documents, such as blog posts or news articles, the item profile might include the author, date of creation, and significant words relevant to the document's topic which are extracted from the document. To determine the significance of words in a document, common words with little meaning are first removed. Then, for the remaining words, the *tf*idf* (term-frequency times inverse document frequency) weight is calculated. Words with a *tf*idf* score above a specified threshold t are selected as keywords to be included in the document's profile [3]. The item profile can be represented as a vector of 0's and 1's, where each element indicates whether a specific feature applies to the item (1 for yes, 0 for no). For example, if one element represents the involvement of a particular director, a '1' would indicate that the director directed the movie, and a '0' would indicate they did not. This binary encoding works well for categorical features, but numerical features, such as the length of a movie, require a different approach. For these, we need to apply a normalization technique that scales the values appropriately, allowing them to be integrated into the vector effectively.

User profile is a vector that contains the same components as the item profile; however, the elements are derived differently. The user profile is based on the historical interactions of the user with items, which could be either explicit or implicit. The user profile is then essentially an aggregation of the item profiles with which the user has interacted. If interactions are represented implicitly (with a '1' in the utility matrix), then a simple technique to aggregate these item profiles into a user profile is to calculate the average of the vectors representing the interacted items [4].

Finally, the system recommends an item to a user based on the *similarity* between the user's profile vector and an item's profile vector. This similarity can be calculated using cosine similarity, which measures the cosine of the angle between two vectors. The *cosine similarity* is given by the formula:

$$\cos(\theta) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}$$

In this formula, $\mathbf{u}$ and $\mathbf{v}$ represent the user and item profile vectors, respectively. The value of $\cos(\theta)$ ranges from -1 to 1. Vectors with a cosine similarity close to 1 are very similar, indicating a small angle between them. Conversely, a cosine similarity close to 0 indicates that the vectors are less similar, with an angle close to 90 degrees and completely different with values close to -1.

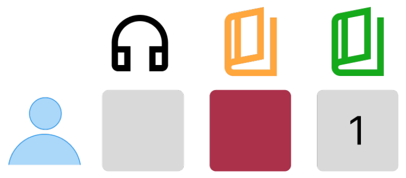

Figure 1.2: This illustration demonstrates content-based filtering, where a user is recommended items similar to those they have previously interacted with. In this example, the blue user has interacted with a green book. Within the system, there are also an orange book and headphones available. Because the orange book is more similar to the green book, it is recommended to the user over the headphones.

1.1.3 Collaborative Filtering-Based Recommendation

A recommender system that uses collaborative filtering (CF) focuses on identifying users with similar behaviors. Unlike content-based filtering models, which focus on the features or properties of items, collaborative filtering models concentrate on the patterns of ratings or interactions among users. This approach assumes that users who share similar tastes will likely interact with similar items. The similarity of two users is based solely on the similarity of their previous interactions in the system.

Memory-based collaborative filtering directly uses the entire user-item interaction matrix to generate recommendations. The system begins by calculating similarities between all user pairs based on their interaction histories. These similarities can be determined using statistical metrics such as cosine similarity. For each user, the system identifies the k-nearest neighbors based on these similarities. Once this set of similar users is retrieved, the system predicts ratings for items that the user has not yet interacted with. Predictions are typically aggregated from the ratings given by the users in this set, and the items with the highest aggregated ratings are recommended. The advantages of this approach include its relatively simple implementation and the ease with which it can accommodate new data. However, there are significant drawbacks in situations of data sparsity, which can lead to the curse of dimensionality [5].

Model-based collaborative filtering involves constructing a machine learning model to understand the underlying relationships in user-item interaction data. Techniques such as matrix factorization—specifically, Singular Value Decomposition (SVD) or Alternating Least Squares (ALS)—are used to decompose the utility matrix into two smaller matrices, assuming that there exists a feature space of smaller dimension that captures the features of users and that of items. Additional methods might include neural networks or clustering algorithms. These models first learn patterns in the data and the trained model is then used to predict how a user might rate or interact with items they have not yet encountered. The advantages of this approach include its capability to handle sparse data, scalability to large datasets, and generally more accurate predictions compared to memory-based methods. The drawbacks include the complexity of implementation and the significant computational resources required for training and tuning the models [5].

A major challenge in collaborative filtering-based systems is the *cold-start problem*. This issue arises because the system relies solely on user interactions; therefore, it cannot make recommendations for a completely new user who has not yet had any interactions. One potential solution to this problem involves combining collaborative filtering techniques with content-based approaches. If any information is available about the new user (e.g., age, country), this data can be used to identify users with similar features. Recommendations can then be made based on the preferences of these similar users, effectively mitigating the cold-start problem. Alternatively, as a simpler but less personalized approach, the system could initially recommend random items to the new user. Even though this method may be less effective at reflecting the user's actual preferences, it allows for immediate interaction opportunities.

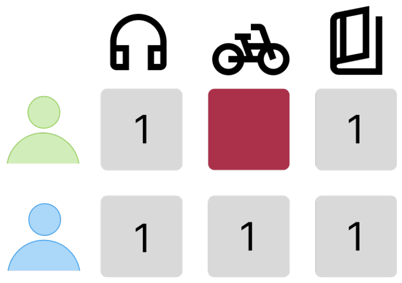

Figure 1.3: This illustration depicts collaborative filtering, where a user is recommended an item based on the similarity of their interactions with those of another user. In this scenario, the green user is recommended a bike because their interactions with headphones and a book closely mirror those of the blue user, who has also interacted with a bike. Consequently, the bike is suggested to the green user.

1.1.4 Hybrid Recommendation

Hybrid recommender systems combine multiple recommendation techniques to address the limitations of individual models. For example, collaborative filtering struggles to make recommendations for a new user due to missing historical interactions of that user, and content-based methods require detailed item attributes, which can lead to an overly large feature space. Hybrid systems merge these approaches to enhance recommendation accuracy and effectively address challenges such as the cold start problem.

1.1.5 Group Recommendation

In many scenarios, there is a need to recommend items to a *group of users*, especially when a decision that requires consensus is involved. For example, whether a family is deciding on their next vacation destination, a group of friends is choosing where to eat, or a fitness studio is selecting music for gymgoers, it is common for the preferences of group members to vary. The goal of group recommender systems is to provide recommendations that maximize the satisfaction of all members of the group [6].

As outlined by the authors in [7], there are several distinct types of groups, each with its own dynamics and recommendation needs:

1. **Established Group**: Consists of individuals who explicitly decide to join together based on shared, long-term interests. For example, a book club, family or a group of friends.

2. **Occasional Group**: Consists of individuals who come together for a specific, often one-time activity, such as a group visiting a museum or a concert. The common thing here is a shared goal at a particular moment.

3. **Random Group**: Involves individuals who share an environment temporarily but have no explicit linked interests, such as visitors in a park, or passengers on a train.

4. **Automatically Identified Group**: These groups are formed by systems that detect clusters of users with similar preferences or needs, often using data-driven algorithms to assess compatibility or shared interests.

The primary challenge for the system is identifying the user groups. Often, the system recognizes existing groups when users explicitly form them by joining entities within the system, such as a running club. However, there are cases where the system does not have explicit information about user relationships and must therefore infer it from the existing data to form groups. This can be achieved by applying k-means clustering directly to the interaction vectors within the utility matrix. If additional user properties are available, they can also be used in the clustering process. In [8], the authors use matrix factorization techniques to find latent features of users. These features are then used to cluster the user vectors in the latent space and the group profiles are created [6].

The next challenge is that of creating the recommendations. There are generally two strategies for aggregating recommendations for a group of users. The first strategy involves generating individual recommendations for each group member and then combining these into a final group recommendation. In this approach, the recommendation process precedes aggregation. The system may either generate a set of candidate items from which users can collectively choose without system-generated ranking, or it can create individual predictions for each member and then aggregate these into a final recommendation. The second strategy begins by aggregating the preferences or interactions of all members of the group to form a *group profile*, which the system treats as a single user. Recommendations are then generated based on this group profile, placing the aggregation step before the recommendation process [9].

The main challenge in all mentioned group recommendation scenarios is tailoring the recommendation to the group as a whole, given information about the individual preferences of group members. Since there is no optimal way to aggregate recommendation lists, various aggregation functions are used to come up with recommendations that consider individual preferences as much as possible.

Aggregation functions can be categorized into three types: majority-based, consensus-based, and borderline [9]. Below is an overview of different types of aggregation functions taken from the social choice theory [10] and their categorization into one of these categories:

1. **Majority-based aggregation functions**: These functions focus on items that are most popular among the group.

   - **Plurality Voting**: Identifies the item that receives the highest number of votes.
   - **Borda Count**: Prioritizes the item with the best total ranking score, calculated by assigning a score from 0 to $n_{items} - 1$ to each item rank. Scores for equally evaluated items are averaged.
   - **Copeland Rule**: Selects the item that performs best in pairwise evaluation comparisons with other items.
   - **Approval Voting**: Recommends items that receive the most support from the users, measured by the number of item evaluations above a defined threshold.

2. **Consensus-based Aggregation Functions**: These functions consider the preferences of all group members.

   - **Additive Utilitarian**: Chooses the item with the highest total of user-individual evaluations.
   - **Average**: Selects the item with the highest average of user-individual evaluations, which can be problematic in larger groups as each individual's opinion carries less weight.
   - **Multiplicative**: Picks the item with the highest product of user-individual evaluations.

  - **Average without Misery**: Recommends items that achieve an average rating above a specified threshold, excluding those with any individual ratings below that threshold.
  - **Fairness**: Assigns rankings as if individuals were taking turns selecting items.

3. **Borderline Aggregation Functions**: These functions focus on a subset of the user preferences.

    - **Least Misery**: Chooses the item that receives the highest of the lowest evaluations given by any group member.
    - **Most Pleasure**: Selects the item that receives the highest evaluations from individuals.
    - **Majority Voting**: Identifies the item that garners the majority of evaluations across the group.
    - **Most Respected Person**: Recommends items based on the evaluation proposed by the most respected or influential group member.

These aggregation functions are used to aggregate the final predictions into either a ranked list of candidate items or a single aggregated group recommendation. Both collaborative (see Section 1.1.3) and content-based (see Section 1.1.2) filtering approaches are employed for group recommendations.

1.1.6 Point-of-Interest Recommendation

Point-of-Interest (POI) recommendation focuses on suggesting new unvisited places such as museums, restaurants, and public spaces to users within the system. Most current research data is derived from Location-Based Social Networks (LBSNs), which is a specific type of online platform where users can log their visits to various locations through *check-in*s which represent implicit feedback. Google Places, Yelp, Foursquare or Gowalla are examples of LBSNs. The primary goal of the POI recommendation systems is to reduce information and choice overload and still provide tailored recommendations to each user.

POI recommendations are very similar to classic recommendation systems; however, they require more specific data beyond the usual user-item interaction matrix. Most importantly, geographical information, such as the coordinates of check-ins, plays a large role in POI recommendations. In addition, these systems often incorporate constraints like opening and closing times, pricing, and other factors relevant to POI to improve the relevance of their suggestions [11].

In POI recommendation systems, the collected check-ins typically consist of timestamps that indicate when a user visited a point of interest. Additionally, users may visit the same POI multiple times, which is a scenario that does not occur in traditional user-item matrices. To accommodate this, frequency matrices are constructed that count the number of times a user visits each POI [11].

The general optimization problem for recommendations described in Equation 1.1 can be adapted to better suit point-of-interest recommendations [11]:

$$l^*(u) = \arg\max_{l \in \mathcal{L}} g(u, l, \theta) \tag{1.2}$$

In this model, $l^*(u)$ represents the optimal POI for user u, selected from the set $\mathcal{L}$ of all available POIs. The function g models the relevance of each POI l for user u in the context θ, which includes variables such as the coordinates of users and POIs, along with opening and closing times.

The algorithms used in POI recommender systems can be classified into the following approaches [11]:

1. **Based on Similarities:** These algorithms use classic k-NN approaches that utilize user or item similarities, such as cosine similarity. Some systems leverage social networks, using a user's friends as "nearest neighbors" to find shared interests for more personalized recommendations. Neighborhood-based models are often combined with other recommendation strategies. An example is the LARS (Location-Aware Recommender System) model [12], which considers travel locality —assuming users prefer not to travel far — and preference locality — assuming that users from the same region tend to have similar preferences.

2. **Factorization Models:** These algorithms aim to decompose the check-in matrix into two latent space matrices, one for users and one for locations, while several works are also incorporating geographical, temporal, or social influences. An example of such an algorithm is GeoMF [13], a weighted matrix factorization model that defines the areas of user activity and POI influence within subdivided spatial grids. Another example is RankGeoFM [14], where the authors incorporate neighboring POIs of the target POI and include the influence of their distances in the optimization process.

3. **Deep Learning:** Since 2020, deep learning has become the dominant model type in POI recommendation. Various neural network architectures are employed, including:

   - **Multilayer Perceptrons (MLP)**: The simplest form of neural networks used in this domain.
   - **Autoencoders**: These models learn by encoding information into a latent space and then decoding it back to reconstruct the input.
   - **Convolutional Neural Networks (CNNs)**: Primarily used for processing images.

- **Recurrent Neural Networks (RNNs)**: Ideal for memorizing previous computations and processing sequential information.
- **Word / Graph Embeddings**: Techniques that learn latent representations of words / graphs or similar structures, often using matrix factorization.

A notable example is the PACE (Preference And Context Embedding) algorithm [15], which learns separate embeddings for users and POIs. These embeddings are then merged and fed into a neural network to predict user preferences and the context associated with users and POIs.

4. Other approaches include the **Probabilistic**, **Graph-Based** or **Link-Based** and **Hybrid**.

1.2 Location Encoding

In order to recommend item categories for users in a specified area, we need to be able to represent (or *encode*) location coordinates into an embedding so that it can be used by a machine learning model.

There are many tasks for encoding geolocation data such as polylines, polygons, and graphs (e.g., trajectories, administrative regions, and transportation networks respectively); however, in this work, we focus solely on encoding points (location coordinates). Therefore, the type of encoder we discuss is referred to as the *single-point location encoder*, denoted by $Enc(\mathbf{x})$.

In [16], the authors define the location encoder in general as a function

$$Enc^{(\theta)}(\mathbf{x}) : \mathbb{R}^2 \to \mathbb{R}^d$$

where $\mathbf{x} \in \mathbb{R}^2$ represents a 2D location coordinate, d is the dimension of the latent vector space into which the location is being encoded, and θ are the parameters of the model function. The resulting encoded vectors are referred to as *location embeddings.* In [16], the definition is more broadly stated to also include the dependency of the function on other geolocation data and working with 3D spatial coordinates, but here we have narrowed the definition to be most relevant to this work. The concept of the location encoder encompasses not only the modeling of location data but also its application to downstream tasks, essentially representing the entire model from input to output for the task at hand.

The objective of a location encoding model is to learn the non-linear conditional probability distribution of observing the true values defined by the downstream tasks given the input of location $\mathbf{x}$.

1.2.1 Properties of Location Encoder

According to the definition given in [16], the location encoder must conform to a set of criteria, including preserving distance and being direction-aware. A location encoder that only preserves distances but ignores changes in direction is referred to as *isotropic* location encoder. Additionally, the encoder should comply with other properties such as the capability for inductive learning, task independence, and fixed parametrization with a finite set of parameters. We further describe these notions presented in [16]:

1. **Distance Preservation** property ensures that two geographically close location are be embedded into two similar location embeddings. The dot product of the two embeddings of two locations must therefore decrease as the physical distance between the two locations increases. This principle asserts that the nearer two locations are in physical space, the more similar their corresponding embeddings will be.

2. **Direction Awareness** property ensures that two locations pointing in similar directions have more similar embeddings compared to those pointing in different directions. Empirical studies hint at some multi-scale encoders being sensitive to direction, yet there is minimal theoretical support or intentional efforts to develop encoders specifically for recognizing direction. While the location encoders often preserve distance well, the intentional inclusion of directional information is still relatively unexplored in research.

3. **Inductive Learning** property ensures that the trained location encoder can process any location $mathbfx$, even those not previously encountered during training. Therefore, the encoder does not require retraining when new "unknown" locations are added to the dataset or the encoder is requested to encode them.

4. **Task Independence** property requires that the architecture of the encoder can be applied to any downstream task without modifying the architecture, ensuring its reusability.

5. **Fixed Parametrization** property constrains the complexity of the model by requiring it to have a fixed set of parameters, regardless of the dataset size. While this model lacks flexibility, it benefits from maintaining consistent complexity.

1.2.2 Types of Location Encoder

The common structure of a *single-point location encoder* can be described as follows [16]:

$$Enc(\mathbf{x}) = \mathbf{NN}(PE(\mathbf{x}))$$

where $PE(\mathbf{x}) \in R^W$ is typically a deterministic function called the *positional encoder*, which encodes the coordinate $mathbfx$ into a position embedding vector. $\mathbf{NN}(\cdot)$, the neural network component of the encoder, contains learnable parameters that are trained to model the specific task at hand.

Based on $PE(\cdot)$, the single-point location encoder is classified into categories of discretization-based, direct, sinusoidal, and multi-scale. All these encoders use different techniques to encode the initial position $\mathbf{x}$. Discretization-based and direct encoders are the most relevant to this work.

- **Discretization-Based Location Encoder** partitions the geographical area into discrete units of a given shape, which are then encoded. For example in [17], the authors use one-hot encoded vectors of the size, corresponding to the number of discrete areas. Each coordinate $\mathbf{x}$ is represented by a vector where a '1' is placed at the ith position if the coordinate falls into the ith discretized area. This approach faces several challenges: each discretized area embedding is trained separately, which does not preserve the inductive learning property; density of location points might vary in the discretized area, which can be addressed by applying finer discretization in areas with a higher density of points; and the challenge of preserving distance, which can be managed by adding regularization.

- **Direct location encoder** typically involves normalizing or standardizing the input location feature x before processing it through a neural network. Various implementations of direct location encoders include normalizing coordinates to a range of $[-1, 1]$ (e.g., [18]) or employ other scaling techniques. Although this method is straightforward, it fails to capture the spatial relationships between locations without the use of embeddings and feature decomposition [16].

In this work we focus on integrating Location Embedding 1.2 techniques with Recommender Systems techniques 1.1.

Chapter 2

Data

In this study, we utilize the publicly available datasets from four major U.S. cities: **Chicago, Illinois**; **Philadelphia, Pennsylvania**; **Austin, Texas**; and **San Francisco, California**. These datasets contain Google Local Data dated up to September 2021. The datasets were sourced from the University of California, San Diego McAuley Group Data Repository [19, 20, 21], and are essential for the development of our location-based recommender system. Each dataset consists of user-generated reviews, which provide insights into consumer preferences and behavior, and business metadata, which contains information on business locations, categories, operational hours, and other attributes.

However, the raw datasets include data for the entire states of Illinois, Pennsylvania, Texas, and California; therefore, further processing is required to isolate data related exclusively to the respective cities and to our problem.

2.1 Business Metadata Dataset

For each state, we sourced a dataset [19] containing business metadata of businesses in Google Maps. This dataset is provided as a *JSON* file, where each line represents a *JSON* object containing metadata for a single business. Below is a brief overview of the key object fields included in this dataset:

Table 2.1: Overview of Fields in the Business Metadata Dataset.

Field Name	Description
`gmap_id`	Unique identifier for the business
`latitude`	Geographical latitude of the business
`longitude`	Geographical longitude of the business
`category`	List of categories of the business
`price`	Price level of the business

2.2 User Reviews Dataset

For each state, we sourced a dataset [19] containing user reviews for businesses in Google Maps. This dataset is provided as a JSON file, where each line represents a JSON object containing a single user review of a business. Below is a brief overview of the key object fields included in this dataset:

Table 2.2: Overview of Fields in the User Reviews Dataset

Field Name	Description
`user_id`	Identifier for the user
`gmap_id`	Identifier for the business
`rating`	Rating given by the user (1-5)
`text`	Content of the review

2.3 Geospatial Data

We use **geographic shapefiles** to precisely define and analyze the cities of **Chicago, Illinois**; **Philadelphia, Pennsylvania**; **Austin, Texas**; and **San Francisco, California**. These shapefiles represent the whole cities which are split into polygons representing their respective neighborhoods. In this work the shapefiles are used for filtering data, visualization and mapping.

2.3.1 Geographic Filtering of Data

Given that the original datasets consist of business data from entire states, it is necessary to refine this data to include only those businesses located within the city limits that are defined by the shapefiles. This filtering ensures that our analysis is specific to the required urban areas.

2.3.2 Visualization and Mapping

Furthermore, these shapefiles are vital for the visualization and mapping aspects of our study. They provide well-defined neighborhood boundaries that enable us to plot various pieces of information on maps, allowing for a detailed spatial analysis of patterns and trends across different neighborhoods within the cities.

2.3.3 Sources of Shapefiles

The neighborhood shapefiles used in this study are sourced from official geographic information system (GIS) data portals for each respective city, providing publicly accessible and up-to-date geospatial data. Specifically, the shapefiles for Philadelphia are obtained from OpenDataPhilly [22], those for

San Francisco from the San Francisco Data Portal [23], the Austin shapefiles from the Austin Public Data Portal [24], and the Chicago shapefiles from the Chicago Data Portal [25].

2.4 Data Processing

This section outlines the various stages of data processing that are applied to prepare the dataset for subsequent analysis.

2.4.1 Geospatial Filtering

The original dataset, comprising business metadata (see Section 2.1), contains businesses from various places in the states of Illinois, Pennsylvania, Texas, and California. However, for the purposes of this study, our analysis is specifically focused on the cities of Chicago, Philadelphia, Austin, and San Francisco. Therefore, the initial processing step involves filtering businesses based on whether their geographical coordinates (`latitude`, `longitude`) fall within the polygons defined by the respective shapefiles of the cities. In practice this geospatial filtering is performed using the Python package *geopandas* [26].

2.4.2 Restaurants Filtering

The original business metadata dataset includes businesses across a wide range of categories (e.g., Pharmacy, Fire Station, Restaurant, ...), where individual businesses may be associated with multiple categories in the `category` field. In this study, our focus is strictly on **restaurants**; hence, it is necessary to refine the dataset to include only businesses that fall under this classification. Several methods could be used to define whether a business is categorically a restaurant were explored, including an analysis of the business's title and textual reviews. However, for simplification reasons, the adopted approach involves a straightforward examination of the `category` field. Initially, all category strings are converted to lowercase to ensure case-insensitive comparisons. Subsequently, businesses are retained if and only if any entry in their `category` field contains the substring 'restaurant'. We note that not all dining establishments in the dataset would necessarily have this word present.

2.4.3 Categories Filtering

From these businesses all **unique categories** along with their frequencies are extracted, those occurring more than $d = 30$ times are identified as a set of **relevant categories**. For each business we retained only those categories that were part of this frequently occurring set. Following this categorization refinement, an additional filtering step is applied. Businesses that are left with no category or with only the generic category of 'restaurant' are removed from the dataset. This step is taken because such a general category does not provide enough detail to analyze the diverse types of restaurants intended for this study. Table 2.3 illustrates which businesses are kept ('Keep') or discarded ('Discard') based on their categorization.

Table 2.3: Examples of filtering businesses based on categories.

`gmap_id`	`category`	**Action**
0x89c...	italian restaurant	Keep
0x67a...	pharmacy, drug store	Discard
0x34c...	fire station	Discard
0xa31...	restaurant; bar	Keep
0xbcc...	bakery; cafe	Discard
0x57e...	restaurant	Discard

2.4.4 *k*-core Decomposition

The process of *k*-core decomposition is used to analyze the structure of a network by peeling off the outermost layers, progressively revealing denser, more connected cores. It is particularly useful in social networks and recommender systems as it helps to identify communities or clusters of users who share common interests or interactions. [27]

Definition 2.4.1 (k-core Decomposition)**.** Given an undirected and unweighted graph represented as $G(V, E)$ where V is the set of vertices and E is the set of edges and given a threshold value k, the k-core decomposition of G is the process of systematically removing vertices from G to create $G'(V', E')$ (a subgraph of G) where each vertex $v \in V'$ satisfies the condition $\deg(v) \geq k$. A vertex v is removed from the process if it does not satisfy the condition $\deg(v) \geq k$.[27]

In the context of recommender systems the relationships between *users* and *items* can be represented as a *bipartite* graph $G(V_u, V_i, E)$ where the V_u (users) and V_i (items) are disjoint sets and every edge $e \in E$ connects a vertex in V_u to a vertex in V_i. Figure 2.1 shows an example of the k-core decomposition of the user-item relationships graph where red resp. green nodes represent users resp. items and edges represent an interaction.

We continue to process our dataset by merging the user reviews dataset (2.2) with the refined business metadata dataset using the `gmap_id` field as a key to get a resulting dataset of reviews merged with the restaurants and their categories created in the previous steps. The k-core subset is then extracted for $k = 5$, ensuring that in the resulting dataset, each user has rated at least k restaurants, and each restaurant has been rated by at least k users.

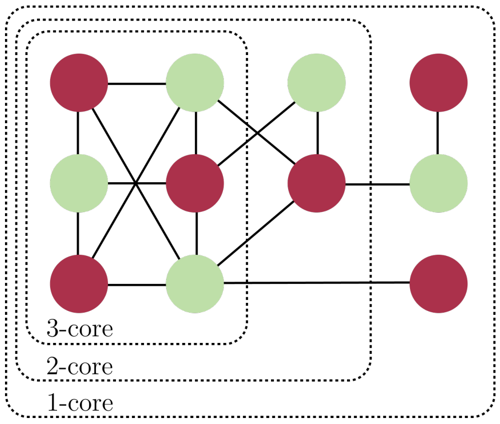

Figure 2.1: Example of the k-core decomposition of a bipartite graph.

This step not only focuses the analysis on a denser core where interactions are more frequent but also helps in mitigating the effects of outliers, as it removes nodes with connections fewer than k. By employing the k-core decomposition, we enhance collaborative filtering techniques, allowing the recommendations to be based on more reliable data. This method is likely to increase the accuracy of the system's suggestions by concentrating on the most significant and consistent user-item interactions.[27]

2.4.5 User Location Estimation

In our dataset, direct user location data is not available; instead, we have access to locations of the restaurants where users have checked in.

A straightforward initial approach to estimate a user's "home" location might be calculating the arithmetic mean of all restaurant locations they have visited. However, this method tends to produce resulting locations that are too centered within the entire area, often suggesting that a user's home location is near the geographic center of the map, as shown in Figure 2.2(b).

This centering tendency fails to reflect the realistic distribution of residential locations. For instance, using this method, the likelihood of identifying residences at the periphery of the city is notably low, and it might inaccurately place a user's location in non-residential areas like at an airport or within a large park. Importantly, multiple studies in the literature suggest that individuals tend to visit POIs closer to their homes, venturing farther only on fewer occasions [28, 29]. This behavioral pattern implies that a user's most frequently visited area is a more reliable indicator of their home location than a simple arithmetic mean of all visited locations.

Therefore, we selected a more principled method for estimating user locations by leveraging the geospatial division of cities into neighborhoods provided by the prepared shapefiles (Section 2.4.1). We calculate the average location of check-ins within the neighborhood where a user most frequently checks in. This approach respects the spatial distribution of user check-ins, the realistic distribution of residential locations within the city, and aligns with observed behaviors regarding proximity and travel distances.

For calculating the average location, we choose the method of directly calculating the mean of latitude and longitude coordinates. This approach is deemed adequate due to the small scale of the areas—individual cities—where the curvature of the Earth's surface introduces minimal distortion to the geographic calculations. With this simplification, the computational overhead of transforming coordinates into a projected coordinate system can be avoided. Moreover, preliminary analyses confirmed that the results obtained by direct averaging are sufficiently close to those derived from more complex spatial calculations.

Figure 2.2 compares the two user location imputation methods and their resulting distributions within the Philadelphia area. In panel (a) we can see gaps of areas without users, which correspond to non-residential zones such as the airport and industrial sectors in the south, or the Fairmount Park above the center. In contrast, panel (b) demonstrates a user distribution that is denser in the city center and gradually fades towards the borders, reflecting the influence of the averaging method.

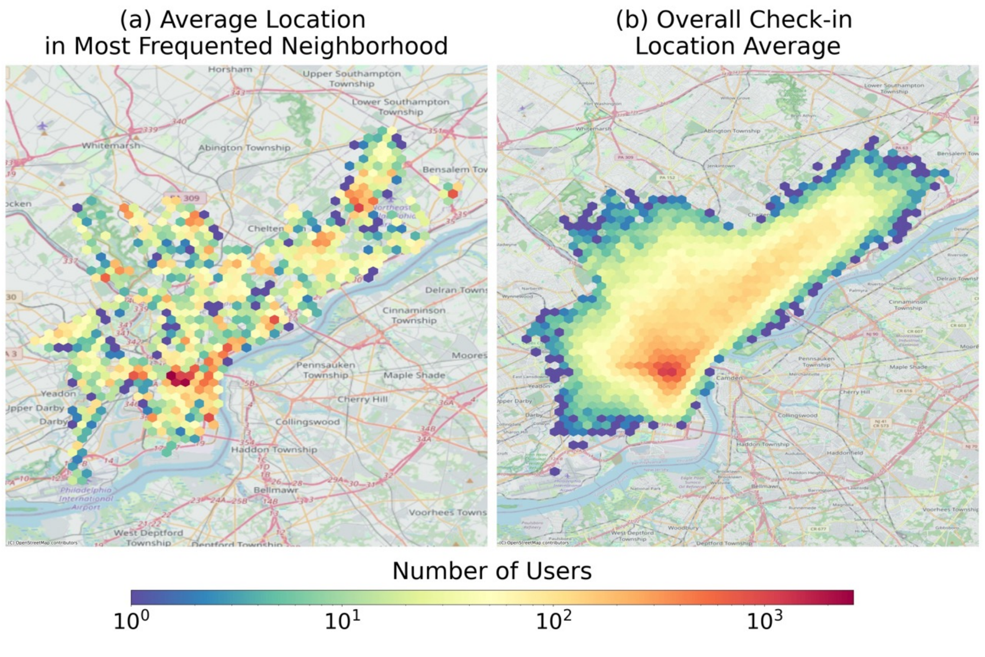
(a) Average Location in Most Frequented Neighborhood
(b) Overall Check-in Location Average
Number of Users
10^0
10^1
10^2
10^3

Figure 2.2: Comparison of methods for imputation of user locations in the city of Philadelphia.

From Figure 2.3 it is also apparent that the method depicted in (a) can identify multiple attractive locations as opposed to (b) where there seems to only be a single “hot” center.

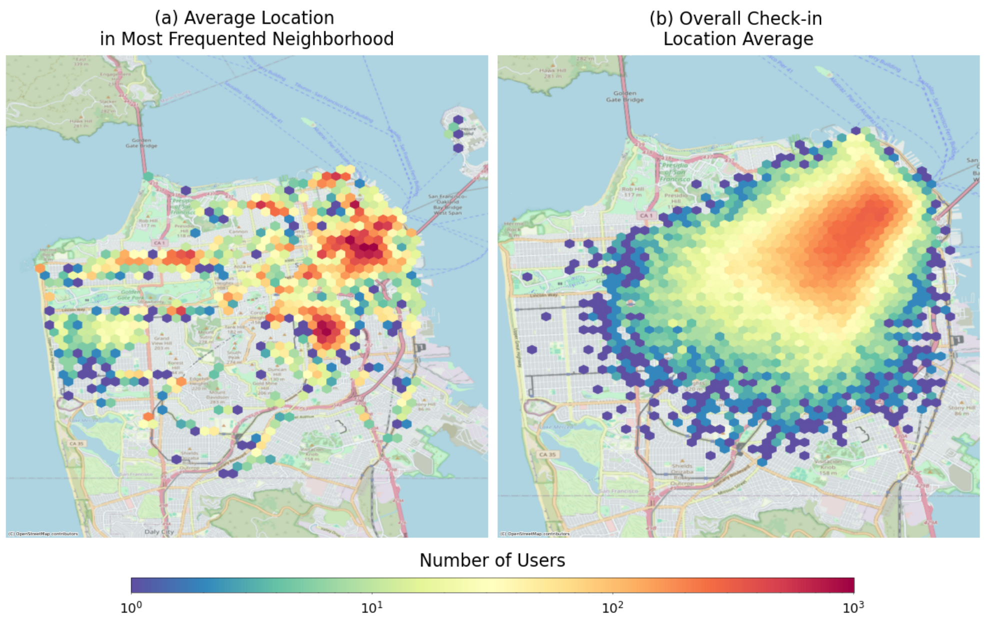

Figure 2.3: Comparison of methods for imputation of user locations in the city of San Francisco.

2.4.6 One-Hot Encoding of Categories

The final step in the data processing pipeline is to perform one-hot encoding on the `category` column. This involves transforming all unique categories into separate columns within the dataset. Each entry is then marked with a 1 in the column corresponding to its category if the category is present, or a 0 if it is not.

2.5 Dataset Overview

Table 2.4: Statistics of the Datasets for All Cities.

City	**# Users**	**# Items**	**# Ratings**	**Sparsity**	**Area (km^2)**
Austin	67,543	2,747	879,868	0.0047	704
Chicago	116,962	6,622	1,461,774	0.0019	606
Philadelphia	49,105	3,311	581,169	0.0036	369
San Francisco	43,614	3,402	517,117	0.0035	121

In Figure 2.4, we observe that the interaction of users with restaurants across various neighborhoods in the cities shows a fairly uniform distribution of categories. Given that Chicago is the largest city, it is reasonable to expect that it includes the largest variety of categories.

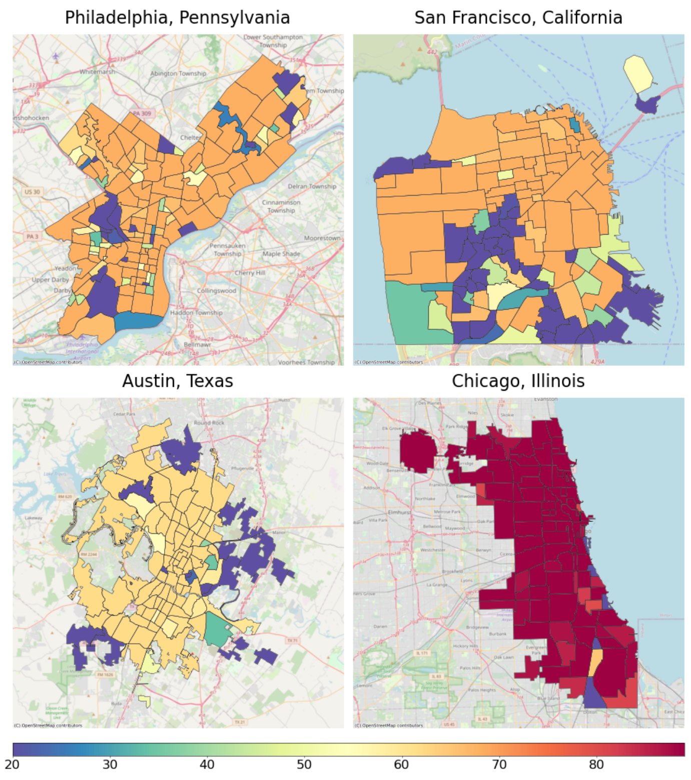

Figure 2.4: Number of unique categories that users from each neighborhood in each city interact with.

Chapter 3

Method

This section details the approach to develop a machine learning model designed to recommend the most suitable categories for a new Point of Interest (POI) based on location data and preferences of users in the neighborhood. This task presents unique challenges due to the spatial dimension of the data, which requires an approach that considers both the popularity of different categories of items (restaurants) and their geographic distribution. In this section we introduce the two problems that the proposed model aims to solve, the architecture of the model and the reasoning behind it.

3.1 Problem Definition

We define a set of triplets $\mathcal{P} = \{p_i | i = 1, \ldots, N\}$. Each triplet p_i represents an interaction of a user rating a POI which is described by $p_i = (\mathbf{x}_i, \mathcal{C}_i, r_i)$. Here, $\mathbf{x}_i \in \mathbb{R}^2$ represents the geographical coordinates of the user, given as $\mathbf{x}_i = (\mathrm{lon}_i, \mathrm{lat}_i)$. The explicit rating $r_i \in \{1, \ldots, 5\}$ is an integer reflecting the user's satisfaction level. The set $\mathcal{C}_i \subseteq \{1, \ldots, C\}$ contains indices of the $|\mathcal{C}_i|$ categories relevant to the rated POI. The set $\{1, \ldots, C\}$ comprises all C unique category indices across the dataset $\mathcal{P}$ and it is effectively the union of $\mathcal{C}_i$.

In this work, we employ solutions to the two following problems:

1. **Category Probability Modeling**: Given the set of points $\mathcal{P}$, we define a function $f_{\mathcal{P},\theta}$ parameterized by θ, which maps input features $\mathbf{x} \in \mathbb{R}^2$ to a vector of probabilities $\hat{\mathbf{p}} \in \mathbb{R}^C$:

$$f_{\mathcal{P},\theta}(\mathbf{x}) : \mathbb{R}^2 \to \mathbb{R}^C \tag{3.1}$$

   This function estimates the probabilities of a user that belongs to a given point x interacts with each of C categories. Note that these probabilities are not constrained to sum to one, reflecting that a user could, for instance, have a 0.8 probability to interact with an Italian restaurant and, at the same time, 0.8 to interact with a Chinese one.

2. **Rating Probability Distribution Modeling**: Given the set of triplets $\mathcal{P}$, we define a function $g_{\mathcal{P},\mu}$ parameterized by μ, which maps input features $\mathbf{x} \in \mathbb{R}^2$ to a matrix $\hat{\mathbf{P}} \in \mathbb{R}^{C\times 6}$. For each $i \in \{1, \dots, C\}$, the row vector $\hat{\mathbf{p}}_i \in \mathbb{R}^6$ of $\hat{\mathbf{P}}$ is a probability vector.

$$g_{\mathcal{P},\mu}(\mathbf{x}) : \mathbb{R}^2 \rightarrow \mathbb{R}^{C\times 6} \tag{3.2}$$

This function estimates for each of the categories $c \in \{1, \dots, C\}$ the probability distribution of c receiving a specific rating. Specifically, each element of $\hat{\mathbf{p}}_i$ corresponds to the probability of obtaining a rating of 1 through 5, or no rating at all (absence of value), which is represented by the first element of each row vector $\hat{\mathbf{p}}_{i,0}$. The first element thus captures the probability that a rating is not available or applicable, and the subsequent elements represent the probabilities of ratings from 1 to 5, respectively. Note that the vectors $\hat{\mathbf{p}}_i$ are probability distributions, and therefore, their values sum to one. This reflects the scenario where, for example, a user might rate an Italian restaurant as follows: 1 with a probability of 0.1, 2 with 0.1, 3 with 0.2, 4 with 0.1, and 5 with 0.1 and with the probability of giving no rating being 0.3.

3.2 Model Architecture

In this section, we describe the proposed architecture of the machine learning model used to solve the problems defined in Section 3.1.

Our architecture is designed to be **task independent** (see Section 1.2.1), enabling application to any problem that requires encoding geographical coordinates into an embedding space. These embeddings can then be seamlessly integrated into downstream neural network modules to tackle various tasks.

As depicted in Figure 3.1, the architecture comprises several key components:

- **Area Discretizer**: Segments the geographical area into discrete zones.
- **Latitude and Longitude Embeddings**: Convert discretized latitude and longitude bins into a dense embedding space, capturing spatial relationships.
- **MLP**: Combines the embeddings to perform the task-specific computations.
- **Transformation Function** ϕ: Transforms the outputs of the neural network into a format suitable for making predictions or further processing.

- **Output Value** $\hat{y}$: Represents the predicted result, tailored to the specific requirements of the task at hand.

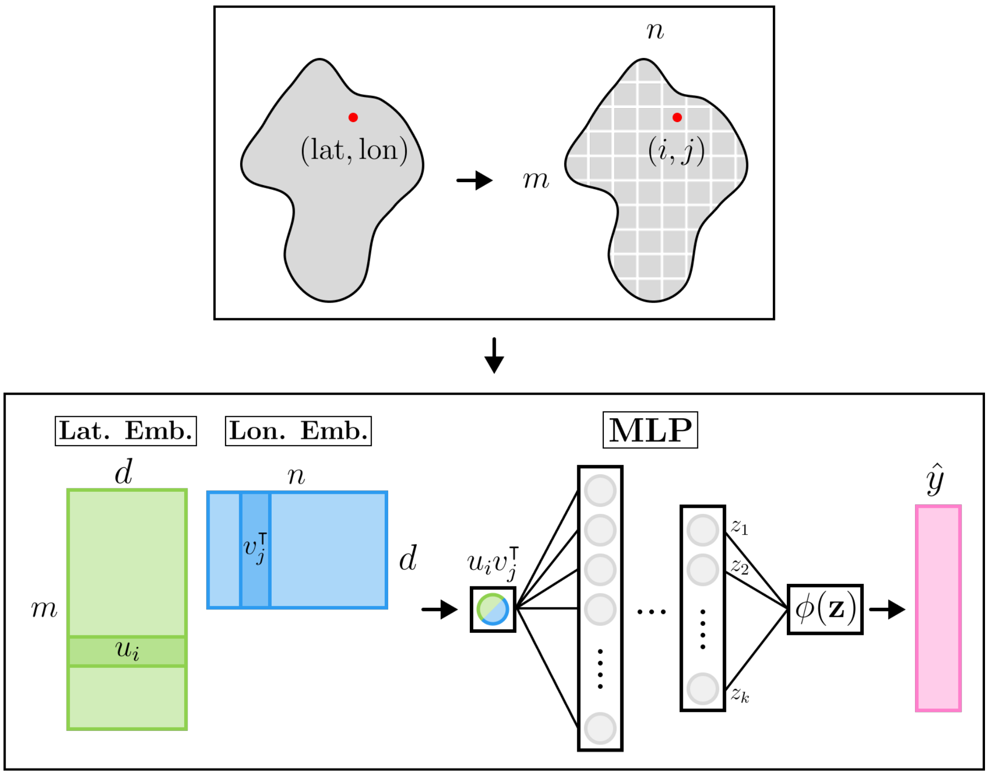

Figure 3.1: Structure of The Proposed Model

In the following sections, we further describe the individual components of the proposed model.

3.2.1 Input

The location coordinate $\mathbf{x} = (\text{latitude}, \text{longitude}) \in \mathbb{R}^2$ is the input to the proposed model. For both the **Category Probability Modeling** and **Rating Probability Distribution Modeling** problems defined in Section 3.1, the input is identical. It is the "home" location of a user presented in Section 2.4.5.

3.2.2 Area Discretization

In this study, we adopt the approach of dividing the given geographical area into discrete units of uniform size based on latitude and longitude. These units are parameterized by the `bin_width` parameter. Given that this research focuses on cities that are relatively small in geographic extent, we can reasonably neglect the effects of the Earth's curvature. Therefore, we divide the area into rectangular grid of m rows and n columns. Each cell extends `bin_width` degrees in latitude and `bin_width` degrees in longitude. Each grid cell is referred to as a "bin", and this grid is indexed by `bin_lon` and `bin_lat`, which represent the longitudinal and latitudinal bins, respectively. This spatial discretization is illustrated in Figure 3.1.

The adoption of area discretization allows us to create user groups based on their location.

For the purpose of transforming a location coordinate $\mathbf{x}$ into the grid coordinate $\mathbf{x}_{\text{bin}} = (\texttt{bin\_lat}, \texttt{bin\_lon})$, we use the class `LatLonDiscretizer` throughout the work. Figure 3.2 shows the discretization of two areas and the partition of users.

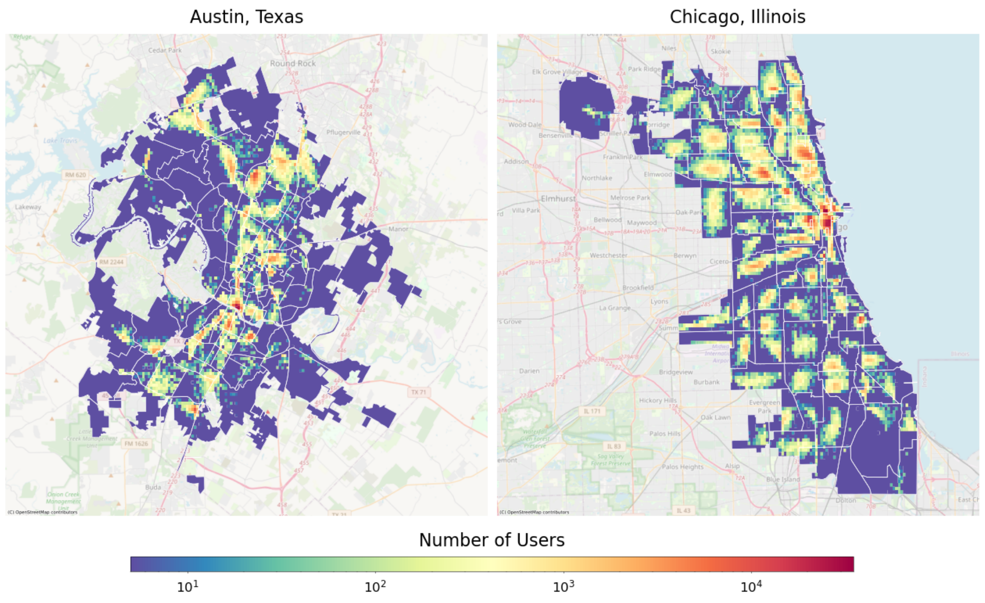

Figure 3.2: Heatmap showing the distribution of users across the *discretized areas* in Austin and Chicago.

3.2.3 Geospatial Embeddings

We encode the discretized areas, representing the preferences of groups of users, into a higher-dimensional feature space that captures the geospatial relationships of the data. As discussed in Section 1.2, the positional encoder $PE(\mathbf{x}_{\text{bin}}) \in \mathbb{R}^W$ is a deterministic function that transforms the input coordinate $\mathbf{x}_{\text{bin}}$ into a vector suitable for learning by the neural network component $NN(\cdot)$. However, in this work, we do not create an embedding for the coordinate as a whole. Instead, we treat $\mathbf{x}_{\text{bin}}$ as two separate entities: `bin_lat` and `bin_lon`. We create separate embeddings for each of the m latitudinal (horizontal) and n longitudinal (vertical) discretized sections, resulting in the embedding matrices $\mathbf{U} \in \mathbb{R}^{m \times d}$ and $\mathbf{V} \in \mathbb{R}^{n \times d}$ which are two fully learnable components and dimension d is the hyper-parameter of the latent space.

The multiplication of the two latent space matrices results in a matrix $\mathbf{H}$ of the same dimensions as the discretized area:

$$\mathbf{H} = \mathbf{U} \cdot \mathbf{V}^{\intercal} \in \mathbb{R}^{m \times n}$$

where we can interpret the element $H_{i,j}$ as the embedding value for the discretized bin (i, j), representing the geospatial embedding of the group of users.

To address the **distance preservation** property discussed in Section 1.2.1, we introduce specific constraints on the vectors within the embedding matrices during optimization. To ensure the latitudinal and longitudinal embedding vectors remain sufficiently close, we employ regularization based on the L_2 norm. Consequently, we augment the optimization loss with the following penalty term:

$$\lambda \left(\sum_{i=1}^{m-1} \|u_i - u_{i+1}\|^2 + \sum_{j=1}^{n-1} \|v_j - v_{j+1}\|^2 \right) \tag{3.3}$$

where u_i (respectively, v_j) is the i-th (respectively, j-th) vector in the matrix $\mathbf{U}$ (respectively, $\mathbf{V}$). This approach ensures that embeddings for geographically closer locations are closer in the embedding space, effectively preserving the distance properties of the physical space within the model.

Inspired by matrix factorization, this approach meets the requirement of **inductive learning** property discussed in Secton 1.2.1. Even when data is not available for some bins in the discretized area, we can infer the embeddings of these bins as long as there is at least one data point in either the horizontal or vertical direction of the bins. Thus, this method enables predictions for locations not included in the training set. This ability to make predictions for previously unknown locations demonstrates the model's value in those areas where data is often sparse. Figure 3.2 displays a heatmap, which is a matrix where each element represents a bin, and the value indicates the number of users in that bin. In particular, the purple areas in Figure 3.2 represent regions without users; however, thanks to the chosen approach, the model can still make predictions even for these areas.

3.2.4 Multilayer Perceptron (MLP)

Given a particular coordinate $\mathbf{x}_{\text{bin}}$, the model multiplies the latitude embedding vector $u_{\text{bin\_lat}}$ with the transpose of the longitude embedding vector $v^T_{\text{bin\_lon}}$, resulting in a scalar value. This scalar serves as the input to the $\mathbf{NN}(\cdot)$ component of the encoder.

We employ a multilayer perceptron architecture for the $\mathbf{NN}(\cdot)$ component of the encoder, consisting of three fully connected layers with sizes 16, 8, and k. The first two layers, of sizes 16 and 8, utilize the ReLU (3.2.5) activation function. The last layer, of size k, employs an activation function appropriate for the specific task at hand. These layers have proven sufficient for the tasks at hand.

ReLU Activation Function

The Rectified Linear Unit (ReLU) is defined as:

$$f(x) = \max(0, x)$$

This function keeps positive inputs unchanged and replaces negative inputs with zero. ReLU introduces nonlinearity to the dense layers, which is essential for learning complex patterns in the data.

3.2.5 Output Layer

The size k of the output layer varies depending on the problems defined in Section 3.1. We define a function $\phi(\mathbf{z})$, a transformation function that converts the raw output $\mathbf{z}$ of the MLP into the desired format, as illustrated in Figure 3.1.

For the problem of **Category Probability Modeling**, the output of function f (see Equation 3.1) is a vector of probabilities $\hat{\mathbf{p}} \in \mathbb{R}^C$, where C represents the number of all unique categories. In this case, the size of the output layer k is C. Since the expected output is a vector of probabilities, the activation function applied to the output layer is the *sigmoid* function. Applying the sigmoid function directly produces the probabilities, thus no further transformation is needed and therefore $\phi(\mathbf{z}) = \sigma(\mathbf{z}) = \hat{\mathbf{p}}$.

Sigmoid Activation Function

The Sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

This function outputs values between 0 and 1. It useful for models where output needs to be interpreted as a probability.

For the problem of **Modeling Rating Probability Distributions**, the output of function g (see Equation 3.2) is a matrix of probability distributions

$\hat{\mathbf{R}} \in \mathbb{R}^{C,6}$, where C represents the total number of unique categories. Here, the size of the output layer k is $C \times 6$, which is then reshaped to the matrix format. This raw output matrix is denoted az z. Since the desired output is a matrix of probability distributions for each category, the activation function applied to each row of the output layer is the *softmax* function. This ensures that the probabilities in each vector sum up to 1, representing valid probability distributions.

Softmax Activation Function

The Softmax function is defined as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j^K e^{z_j}}$$

where z is a vector of inputs to the softmax layer. This function is particularly effective in multi-class classification problems, as it converts logits to probabilities that sum to one [30].

For the final prediction of a rating given to a category c, the matrix $\hat{\mathbf{R}}$ must be transformed because it contains probability distributions in its rows. To each row $\hat{\mathbf{r}}_i$, we apply a function $q(\mathbf{r}) : \mathbb{R}^6 \to \mathbb{R}$, which outputs the final estimated rating. The function q is defined as follows:

$$q(\mathbf{r}) = \frac{\sum_{i=2}^{6} i \times r_i}{\sum_{i=2}^{6} r_i}$$

When applied to the whole matrix $\hat{\mathbf{R}}$, the function q returns a vector $\hat{\mathbf{r}} \in \mathbb{R}^C$ of estimated rating values for each category. We disregard the first element r_1 (corresponding to “no rating”) in the calculations because it does not contribute to the distribution of ratings (1-5).

The transformation $\phi(\mathbf{z})$ is therefore $\phi(\mathbf{z}) = q(\text{softmax}(\mathbf{z}))$ where z is the raw matrix output of the MLP.

3.2.6 Model Parameters

The model’s parameters are divided to trainable parameters, hyper-parameters and task-specific parameters.

Trainable parameters involve the weights of the MLP dense layers and the learnable weights of the latitude ($\mathbf{U}$) and longitude ($\mathbf{V}$) embeddings.

Hyper-parameters of the model are `bin_width`, the size of the latent space of the embeddings d and the regularization parameter λ which defines the influence of the regularization term proposed in Equation 3.3.

Task-specific parameters include the latitude and longitude bounds of the discretized area and the dimensions of the output.

3.3 Optimization

To solve the problems outlined in Section 3.1, we need to choose a suitable loss function for optimization. This choice is critical because it shapes the optimization algorithm's behavior and determines what an optimal solution looks like. In this section, we will first introduce the loss functions used in this study and then discuss how they are applied in the optimization process.

Mathematical formulation of the optimization problem

$$\theta^* = \arg\min_{\theta} L(\theta, y, \hat{y})$$

where:

- θ represents the parameters of the model,
- y are the actual target values,
- $\hat{y}(\theta)$ are the predicted values given by the model, dependent on θ,
- $L(\theta, y, \hat{y})$ is the loss function that measures the discrepancy between the predicted values $\hat{p}$ and the true values y,
- θ^* are the optimal parameters that minimize the loss function over the training dataset.

3.3.1 Categorical Cross-Entropy

The categorical cross-entropy loss (CCE) is designed to measure the performance of a classification model whose output is a probability distribution across multiple classes. The loss compares the predicted probability distribution with the actual distribution, which is typically represented as a one-hot encoded vector. The closer the model's outputs are to the actual distribution, the lower the loss.

The formula of the categorical cross-entropy loss is given by:

$$L(y, \hat{p}) = -\sum_{r=1}^{R} y_r \log(\hat{p}_r)$$

where:

- y is the one-hot encoded true label vector of the data point.
- $\hat{p}$ is the vector of predicted probabilities that the data point belongs to each class.
- R is the total number of classes and therefore $|y| = R$ and $|\hat{p}| = R$.
- y_r and $\hat{p}_r$ are the components of y and $\hat{p}$ for class r, respectively.

We choose the categorical cross-entropy loss function for the task of estimating the **Rating Probability Distribution** for each item category and a group of users at a given location $\mathbf{x}$. This choice is appropriate because the rating vector (corresponding to y) must represent one of the mutually exclusive *rating categories* (absence of value or rating 1, 2, 3, 4, 5) for each item category. Thus, there can be only one true value in the rating vector. Consequently, the output vector (corresponding to $\hat{p}$) forms a probability distribution in which the values sum up to 1, reflecting the exclusive nature of each rating category. For this task, the defined number of rating categories R, is therefore 6.

Given that the true label for this task is a matrix $\mathbf{Y} \in \{0,1\}^{C,6}$, where C is the number of item categories in the dataset, we calculate the categorical cross-entropy across all categories as follows:

$$L(\mathbf{Y}, \hat{\mathbf{P}}) = \frac{1}{C} \sum_{i=1}^{C} L(\mathbf{y}_i, \hat{\mathbf{p}}_i)$$

where:

- $\mathbf{Y}$ is the matrix of true labels, with each row $\mathbf{y}_i$ representing the one-hot encoded labels for the i-th category.
- $\hat{\mathbf{P}}$ is the matrix of predicted probabilities, with $\hat{\mathbf{p}}_i$ corresponding to the predicted probability distribution for the i-th category.
- $L(\mathbf{y}_i, \hat{\mathbf{p}}_i)$ denotes the categorical cross-entropy loss for the i-th category.

3.3.2 Binary Cross-Entropy

The binary cross-entropy (BCE) is used in binary classification models, where each output prediction is expected to be a probability value between 0 and 1. This loss function measures the performance of a classification model by comparing the predicted probability of the target class with the actual class label, which is either 0 or 1. The loss increases as the predicted probability diverges from the actual label, with the goal of minimizing this divergence.

The formula of the categorical cross-entropy loss is given by:

$$L(y, \hat{p}) = -\left(y \log(\hat{p}) + (1-y)\log(1-\hat{p})\right)$$

where:

- y is the true label of the data point, which can be 0 or 1.
- $\hat{p}$ is the predicted probability that the data point belongs to the class with label 1.

For the task of estimating the **Category Probability** for a group of users at a given location $\mathbf{x}$, we need a loss function that effectively measures the difference between the predicted probabilities and the true category labels. Let $\mathbf{y}$ denote the true label vector for a given data point, where $\mathbf{y} \in \{0, 1\}^C$ and each component y_i of the vector $\mathbf{y}$ is a binary indicator. Specifically, $y_i = 1$ if the rated item belongs to the category at position i, and $y_i = 0$ otherwise. Given the binary nature of $\mathbf{y}$ and that each element of the output vector from the function $f_{\mathcal{P},\theta}(\mathbf{x})$ represents the independent probability of a respective category, we choose binary cross-entropy (BCE) loss for this model.

We then average the binary cross-entropy loss across all categories as follows:

$$L(\mathbf{y}, \hat{\mathbf{p}}) = \frac{1}{C} \sum_{i=1}^{C} L(\mathbf{y}_i, \hat{\mathbf{p}}_i)$$

Chapter **4**

Experiment

4.1 Baseline Methods

In this section, we introduce several baseline methods designed to assess and predict user preferences utilizing location-based data. Each method uses different statistical or machine learning techniques to address the problem of predicting user preferences from geographical and categorical data. We provide detailed descriptions of each baseline method's implementation and highlight the specific computational approaches used.

To make direct comparisons and evaluations possible, all implemented baseline models conform to a uniform structure. Specifically, they are designed to accept the same form of input and generate outputs of the same structure. This standardization ensures that the results can be seamlessly integrated into the evaluation metrics and directly compared with the ground truth values.

The baseline models are all initialized with the following parameters:

- `lon_min, lon_max, lat_min, lat_max` – geographical bounds of the area,
- `bin_width` – width and height of the discretized areas,
- `n_categories` – number of categories,

All baseline models internally use the `LatLonDiscretizer` to process the input location coordinate $\mathbf{x} = (\mathrm{lon}, \mathrm{lat})$ into the discretized location coordinate $\mathbf{x}_{\mathrm{bin}} = (\mathrm{lon}_{\mathrm{bin}}, \mathrm{lat}_{\mathrm{bin}})$. The output is then a vector $\hat{\mathbf{c}} \in \mathbb{R}^{\texttt{n\_categories}}$, where each element represents the values estimated for each category. The interpretation of these values varies depending on the baseline method. A value at position i in the estimated vector $\hat{\mathbf{c}}$ can represent the **estimated rating** given to a category i by users within the area defined by $\mathbf{x}_{\mathrm{bin}}$ (i.e, explicit feedback), or it can indicate the **level of confidence** that users within that area favor the category i (i.e, implicit feedback).

The structure of the baseline models is shown in Figure 4.1.

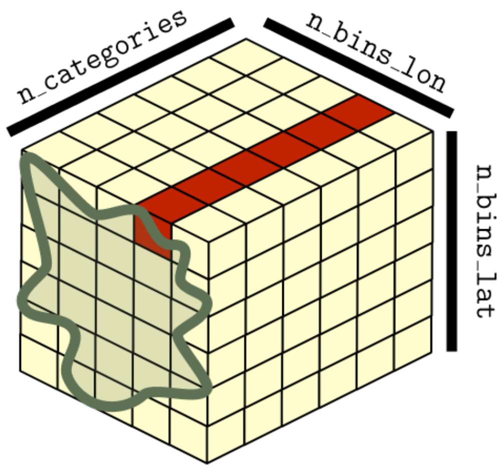

Figure 4.1: The structure of the baseline models using a tensor format. The three-dimensional grid is organized first along two axes, `n_bins_lon` and `n_bins_lat`, which discretize the city area into bins. Each bin has a uniform width and height defined by `bin_width`, segmenting the city into a grid where each cell represents a specific geographic area. The red vector along the `n_categories` axis represents the estimated vector $\hat{\mathbf{c}}$ for users within the area defined by $\mathbf{x}_{\text{bin}}$.

4.1.1 Random (*Rand*)

This baseline serves as an initial benchmark to determine whether the proposed model can outperform a random generation of recommended categories for users at a given location.

For each group of users, defined by discretized areas of the city, this baseline method randomly generates category ratings in the given area.

The input of this model is the location coordinate $\mathbf{x}$ and the output is the vector $\hat{\mathbf{c}}$ of **randomly generated ratings** which are floating-point values ranging from 1 to 5, for all categories at the location $\mathbf{x}$.

4.1.2 Most Popular Categories (*MostPop*)

We choose this baseline as a benchmark to determine whether the proposed model can outperform a method of recommending categories at a given location based on the **Bayesian average** rating given by the users in that area.

For each group of users, defined by discretized areas of the city, we calculate the most popular categories. To determine these categories, we use the

Bayesian average rating for each category within each discretized area. This method aims to provide a more robust evaluation of popularity by incorporating both the average ratings and the number of ratings, and it should provide a more balanced representation of the estimated rating, especially in areas with sparse data. The Bayesian average incorporates prior knowledge about the distribution of ratings into the calculation of an average. The formula for the Bayesian average is as follows [31]:

$$\overline{x} = \frac{C \cdot m + \sum_{i=1}^{n} x_i}{C + n}$$

Where:

- $\overline{x}$ = estimated Bayesian average rating for a category
- C = number of ratings per category
- m = average rating of a category
- n = number of ratings of a category
- $\sum_{i=1}^{n} x_i$ = sum of the ratings for a category

The input of this model is the location coordinate $\mathbf{x}$ and the output is a vector $\hat{\mathbf{c}}$ of the **estimated ratings**, which are floating-point values ranging from 1 to 5, for all categories at the location $\mathbf{x}$.

4.1.3 Matrix Factorization (***MatFac***)

For each group of users, defined by discretized areas of the city, we calculate the **average category confidence vector $\hat{\mathbf{c}}$**. A value in this vector at position i represents the confidence that the particular group of users will like the category i. To determine the category confidence vectors, we use the collaborative filtering method of matrix factorization of the user-item interaction matrix into the latent space matrices $\mathbf{U} \in \mathbb{R}^{m \times d}$ and $\mathbf{V} \in \mathbb{R}^{n \times d}$ where d is the hyperparameter for the size of the latent space, m is the number of users and n is the number of items (restaurants). After estimating the confidence matrix $\hat{\mathbf{R}} = \mathbf{U}\mathbf{V}^\top \in \mathbb{R}^{m \times n}$ we further process it to get the average confidence values of all categories at all locations.

The input of this model is the location coordinate $\mathbf{x}$, and the output is the vector $\hat{\mathbf{c}}$ of **confidence values** for all categories at the location $\mathbf{x}$.

4.1.4 Direct Coordinate Encoder (***Direct***)

In [18], researchers present a model for fine-grained recognition of animal species in images given geolocation data. They begin by normalizing the location vector, $\mathbf{x}$, to the range $[-1, 1]$, resulting in a normalized vector $\mathbf{x}_{norm}$.

This vector is then fed directly into a neural network comprising three fully connected layers with sizes 256, 128, and 128. We adopt this encoder structure as a baseline model for comparison with our proposed model. The objective of this comparison is to determine whether our model, which should capture spatial dependencies, performs better than a model that directly encodes location coordinates. This will help us evaluate whether our proposed method of decomposing location coordinates into embeddings offers any advantages.

4.1.5 Implementation Details

In this section, we detail the implementation of the baseline methods.

Random Model: We implement this baseline as the `RandomModel` class, which is initialized with the parameters specified in 4.1.

A tensor of shape `(n_bins_lon, n_bins_lat, n_categories)` (see Figure 4.1) is created by uniformly generating random ratings, which are floating-point values ranging from 1 to 5. This tensor is static in that it is created only once to ensure that the model, analogous to the proposed trained model, produces consistent outputs for identical inputs.

Most Popular Categories Model: This baseline is implemented as the `MostPopularModel` class, initialized with the parameters detailed in 4.1. It stores the calculated Bayesian average estimations in a tensor with shape `(n_bins_lon, n_bins_lat, n_categories)`, where the last axis contains the estimated ratings for the categories at the specified location coordinate `x`.

Matrix Factorization Model: This model is implemented using the `MatrixFactorizationModel` class, initialized with parameters from 4.1. It utilizes the **alternating least squares (ALS)** algorithm from the Python package *implicit*. Originally designed for implicit feedback, the ALS algorithm is adapted for our needs to process explicit ratings. We train the model solely on samples with ratings of 3 or higher, interpreting these as indicators of positive interactions.

For training, a custom `fit` function is used, involving the following parameters:

- `df` – a *pandas.DataFrame* containing the columns `user_id`, `gmap_id`, `rating`, `user_latitude`, and `user_longitude`, which provide user interactions and location data.

- `factors` – the dimension of the latent space for the ALS matrix decomposition, influencing the granularity of feature representation.

- `regularization` – regularization factor to prevent overfitting in the ALS algorithm.
- `iterations` – number of iterations to perform in the ALS optimization process.
- `confidence_alpha` – scaling factor used to adjust the influence of ratings, enhancing their confidence in the model.

Inside the `fit` function, we construct a sparse user-item matrix $\mathbf{R} \in \mathbb{R}^{m \times n}$, populated exclusively with ratings of 3 or higher, where m denotes the number of users and n denotes the number of items (restaurants). These ratings are treated as confidence levels, indicative of implicit feedback. The entries of $\mathbf{R}$ are scaled by a factor `confidence_alpha` to enhance the distinctions in user preferences. User and item embedding matrices, $\mathbf{U} \in \mathbb{R}^{m \times \texttt{factors}}$ and $\mathbf{V} \in \mathbb{R}^{n \times \texttt{factors}}$, respectively, are then calculated using the ALS algorithm. The recovered matrix $\hat{\mathbf{R}} = \mathbf{U}\mathbf{V}^\top \in \mathbb{R}^{m \times n}$ encapsulates the confidence level of user u_i interacting with item i_j for each indexed pair $(i, j) \in \{1, \ldots, m\} \times \{1, \ldots, n\}$.

The calculated confidences are then translated into the context of categories of users at specific locations as follows:

Each user's "home" location, imputed as described in Section 2.4.5, corresponds to a specific discretized location coordinate $\mathbf{x}_{\text{bin}}$. The rows of matrix $\hat{\mathbf{R}}$, which represent individual users, are grouped according to these discretized locations. An average item confidence vector $\mathbf{c}_g \in \mathbb{R}^n$ is computed for each group. This vector $\mathbf{c}_g$ is stored in the tensor $\mathbf{T} \in \mathbb{R}^{\texttt{n\_bins\_lon} \times \texttt{n\_bins\_lat} \times n}$ at the indices corresponding to each group's discretized coordinates $\mathbf{x}_{\text{bin}}$. For each item (restaurant), a binary vector representing the one-hot encoded categories $\mathbf{b}_j \in \{0, 1\}^{\texttt{n\_categories}}$ is defined as described in Section 2.4.6. Element-wise multiplication of $\mathbf{c}_g$ with each $\mathbf{b}_j$ results in a tensor $\mathbf{T}_{\text{cat}} \in \mathbb{R}^{\texttt{n\_bins\_lon} \times \texttt{n\_bins\_lat} \times n \times \texttt{n\_categories}}$. To aggregate the resulting category confidence vectors, we compute the mean of the non-zero values across the third dimension, yielding a tensor $\mathbf{T}_{\text{final}} \in \mathbb{R}^{\texttt{n\_bins\_lon} \times \texttt{n\_bins\_lat} \times \texttt{n\_categories}}$. An entry at position $(\texttt{bin\_lon}, \texttt{bin\_lat}, i)$ in $\mathbf{T}_{\text{final}}$ represents the average confidence of the category i for users at the discretized location $(\texttt{bin\_lon}, \texttt{bin\_lat})$.

4.2 Evaluation Metrics

In this section, we describe the metrics used to evaluate and compare the effectiveness of the proposed models along with the baseline models presented in the work. The chosen metrics — Mean Reciprocal Rank, Mean Average Precision, and Root Mean Squared Error — provide insights into each model's ranking prediction capability and error rate.

4.2.1 Mean Reciprocal Rank

The mean reciprocal rank (MRR) is a statistical measure used to evaluate the effectiveness of a query response system. It is commonly used in information retrieval and in evaluating recommender systems. Specifically, it focuses on the position of the first relevant item in the list of responses. MRR is mainly useful in scenarios where there exists *one* ideal response to a query.

Definition and formula: Given a set of queries Q with each query $q \in Q$, the Mean Reciprocal Rank is defined as follows:

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i}$$

where rank_i is the position of the first relevant answer for the i-th query within the list of responses returned by the system, $|Q|$ is the total number of queries.

4.2.2 Mean Average Precision

The mean average precision (MAP) is a statistical measure used to evaluate the effectiveness of query response systems. It is commonly used in information retrieval and in evaluating recommender systems. MAP assesses the precision at every relevant position in the list of responses and averages this over all queries. This measure is particularly ideal for situations with *multiple* relevant items per query. It provides a comprehensive indicator of the performance of the system at different levels of recall.

Definition and formula: Given a set of queries Q with each query $q \in Q$, we define **precision at k** (Precision_k) for a query as follows:

$$\text{Precision}_k = \frac{\text{Number of relevant items in the top k positions}}{k}$$

Next, we define the **Average Precision (AP)** for a single query q:

$$\text{AP}_q = \frac{\sum_{k=1}^{n} (\text{Precision}_k \times \text{rel}_k)}{\text{number of relevant items for } q}$$

where n is the number of items retrieved, rel_k is an indicator function equal to 1 if the item at position k is relevant and 0 otherwise.

Finally, the Mean Average Precision is defined as the mean of the Average Precisions for all queries in Q:

$$\text{MAP} = \frac{1}{|Q|} \sum_{q \in Q} \text{AP}_q$$

where $|Q|$ represents the total number of queries.

4.2.3 Root Mean Squared Error

Definition and formula: The root mean squared error (RMSE) is a standard way to measure the error of a model in predicting quantitative data. RMSE quantifies the square root of the average square differences between the predicted and actual values.

Given n observations or predictions, where y_i represents the actual value of the i-th observation and $\hat{y}_i$ denotes the predicted value for the i-th observation, the formula for RMSE is as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

4.3 Model Evaluation

To test the proposed model, we compare its performance with the baseline methods defined in Section 4.1 on the user-restaurant rating dataset prepared in Chapter 2. The model is designed to solve the following tasks formally defined in Section 3.1:

- **Rating Probability Distribution Modeling**: This task involves predicting the probability of each rating value (1-5) for a category c_i at a given location $\mathbf{x}$. Based on these probabilities, we construct a vector of predicted ratings for all categories at location $\mathbf{x}$. This vector is then used to compare with the truth values to calculate the metrics MAP, MRR, and RMSE. Refer to Section 4.2 for detailed explanations on how these comparisons are performed. We label this method as *LLCR*.
- **Category Probability Modeling**: This task predicts the probability that a category c_i is relevant at a given location $\mathbf{x}$ for the vector of all categories. This vector is then used to compare with the truth values to calculate the metrics MAP and MRR. We do not calculate RMSE as this method does not return ratings. Since we aim to retrieve the most relevant categories, we train and validate this model only on such samples from the dataset that have a rating of 3 or higher. We label this method as *LLCC*.

We label the baseline methods as *Rand* for Random (4.1.1), *MostPop* for Most Popular Categories (4.1.2), *MatFac* for Matrix Factorization (4.1.3), *Direct* for Direct Coordinate Encoder (4.1.4).

The evaluation is performed on the datasets of all the cities - Austin, Chicago, Philadelphia, and San Francisco. The datasets are split into training, validation, and test sets in the ratio 70:15:15.

4.3.1 Hyper-Parameter Selection

For each method and city, we conducted a grid search to determine the optimal hyper-parameters, using model performance on the validation set as the criterion for selection. Below, we detail the hyper-parameters considered for each method and identify the best values obtained.

The methods *Rand* and *MostPop* do not involve hyper-parameter tuning due to their deterministic nature.

For the *MatFac* method, the training, validation, and testing datasets were modified to only include ratings with a value of 3 or higher. This adaptation aligns with the model's use of implicit feedback, which requires positive samples for effective evaluation. In the following, we outline the ranges of values tested and the optimal values selected:

- `factors`: Tested values {5, 20, 50, 100}. The optimal value selected for all cities is **5**.
- `regularization`: Tested values {0.1, 0.01, 0.002}. The optimal value for all cities is **0.002**.
- `iterations`: Tested values {50, 100}. The optimal value selected for all cities is **100**.
- `confidence_alpha`: Tested values {1, 50}. The optimal value selected for all cities is **50**.

For the *Direct* method, the only hyper-parameter to be selected is the learning rate of the Adam optimizer. Tested values: $\{10^{-1}, 10^{-3}, 10^{-5}, 10^{-7}\}$. The optimal value selected for all cities is $\mathbf{10^{-5}}$.

The following hyper-parameters are tuned for both the tasks *LLCR* and *LLCC*:

- `bin_width` - Width/height of the discretized area of the city. Tested values: {0.002, 0.003, 0.004, 0.005}. The optimal value is **0.002** for **San Francisco** and **0.003** for all other cities.
- `d` - Size of the latent space for the latitude and longitude embeddings. Tested values: {4, 8, 10, 24}. The optimal value for all cities is **10**.
- `lambda_reg` - Regularization parameter (λ) used to smooth the latitude and longitude embeddings (see Equation 3.3). Tested values: $\{10^{-1}, 10^{-3}, 10^{-5}\}$. The optimal value for all cities is $\mathbf{10^{-1}}$.
- `lr` - Learning rate for the Adam optimizer. Tested values: $\{8\times10^{-5}, 1.5\times10^{-5}, 1.4\times10^{-5}, 10^{-5}\}$. For model *LLCR* the optimal value is $\mathbf{1.4 \times 10^{-5}}$ for **Chicago** and $\mathbf{1.5 \times 10^{-5}}$ for all other cities. For model *LLCC* the optimal value is $\mathbf{3 \times 10^{-5}}$ for all cities.

- `batch_size` - Batch size for training. Tested values: {128, 256}. The optimal value for all cities is **256**.

The graphs in Figure 4.2 indicate that the model *LLCR* tends to start overfitting after a certain number of epochs. This can be seen in the rapid decline of the MAP and MRR curves. However, we can also see that the Loss and RMSE curves for training and validation datasets are already quite converged by the time overfitting becomes apparent.Therefore, we based our decision on the optimal stopping point for model training on the MAP metric. Our analysis suggests selecting the model configuration that achieves the highest MAP, as this metric stabilizes prior to significant overfitting. With this approach, our goal is to maximize the predictive accuracy of our model while avoiding excessive training, which could lead to poor generalization on unseen data. We can also observe that for the cities of Austin and Chicago, the MAP and MRR begin to decline only after approximately 5 epochs, in contrast to the metrics for San Francisco and Philadelphia.

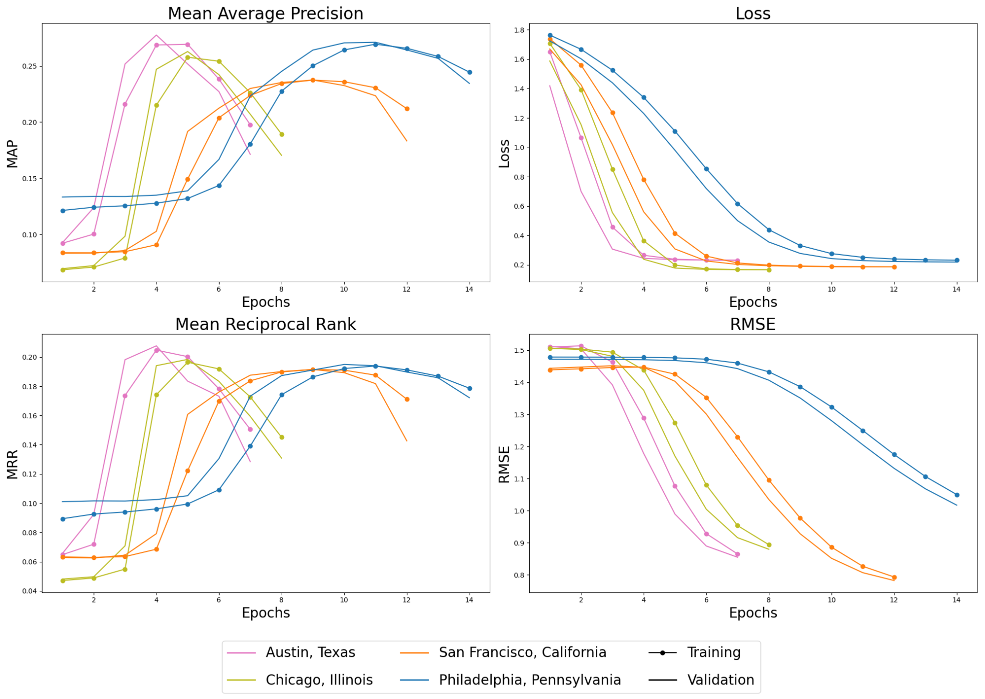

Figure 4.2: Training and validation curves showing the progress of the *LLCR* model's performance metrics (MAP, MRR, RMSE, Categorical Cross Entropy Loss) across epochs, using the best hyper-parameters for Austin, Chicago, San Francisco, and Philadelphia datasets.

On the other hand, in the graphs shown in Figure 4.3, we observe that the model eventually converges for the loss and both metrics. A range of learning rate values was cross-validated; however, in each case, the model converged to the same local optimum.

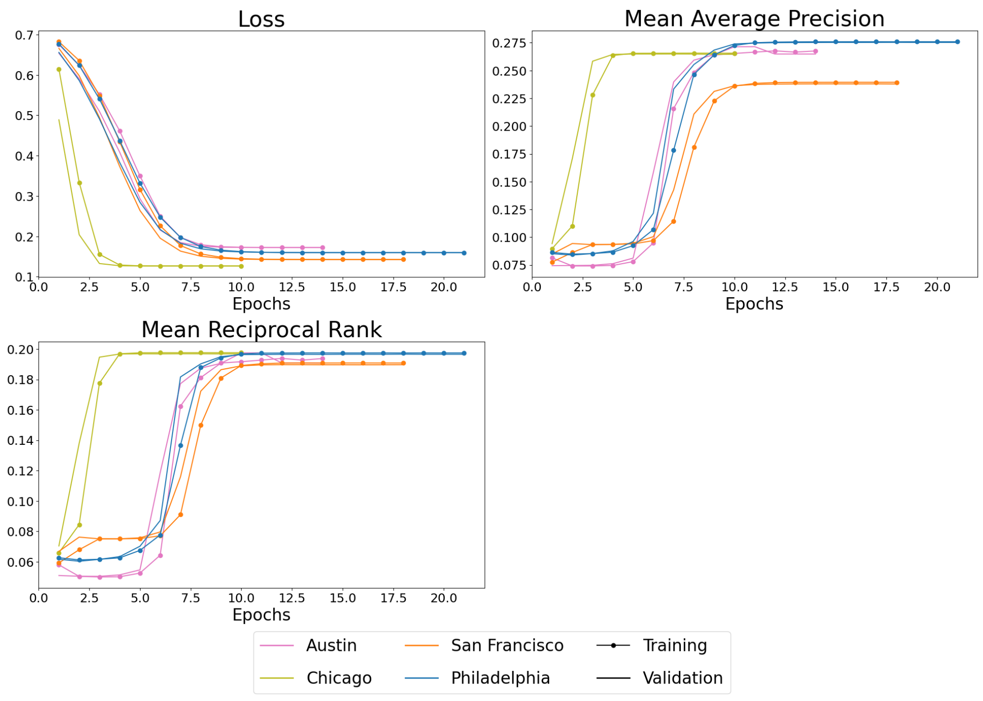

Figure 4.3: Training and validation curves showing the progress of the *LLCC* model's performance metrics (MAP, MRR, Binary Cross Entropy Loss) across epochs, using the best hyper-parameters for Austin, Chicago, San Francisco, and Philadelphia datasets.

4.3.2 Test Data Performance

Table 4.1 displays the performance of all the models with their best hyper-parameters evaluated on the test dataset. We observe that both *LLCR* and *LLCC* models outperform the baseline models. However it is not possible to definitely determine which of the two, *LLCR* or *LLCC*, actually performs better since their results are very close. We can see that the *MostPop* model shows the best performance in terms of RMSE. This is expected since it is based on calculating the average of the ratings in a specific area. Although the proposed models have higher RMSE values, this is not a significant concern for this study, as the primary objective is to maximize recommendation quality, making ranking metrics more important.

In terms of dataset differences, San Francisco had the poorest performance for both *LLCR* and *LLCC*. This suggests that the models found it challenging to uncover the underlying relationships of the preferences of users in this area, which is understandable given the significantly higher user density per km^2 (see Table 2.4) in San Francisco compared to other cities.

Table 4.1: Model Performance Evaluation

		Rand	*MostPop*	*MatFac*	*Direct*	*LLCR*	*LLCC*
Austin	MAP	0.0049	0.0042	0.0067	0.1785	**0.2771**	0.2700
	MRR	0.0033	0.0028	0.0042	0.1351	**0.2071**	0.1958
	RMSE	1.8095	**0.8305**	-	0.8631	1.1781	-
Chicago	MAP	0.0038	0.0039	0.0	0.8208	0.2635	**0.2659**
	MRR	0.0027	0.0028	0.0	0.0572	**0.1988**	0.1979
	RMSE	1.8135	**0.8504**	0.0	0.8551	1.1692	-
Philadelphia	MAP	0.0062	0.0057	0.0100	0.1158	0.2712	**0.2752**
	MRR	0.0042	0.0038	0.0063	0.0873	0.1941	**0.1973**
	RMSE	1.7768	**0.9137**	-	1.4672	1.2027	-
San Francisco	MAP	0.0021	0.0022	0.0026	0.1182	0.2384	**0.2392**
	MRR	0.0015	0.0016	0.0018	0.0971	**0.1921**	0.1912
	RMSE	1.7622	**0.7535**	-	1.4270	0.9237	-

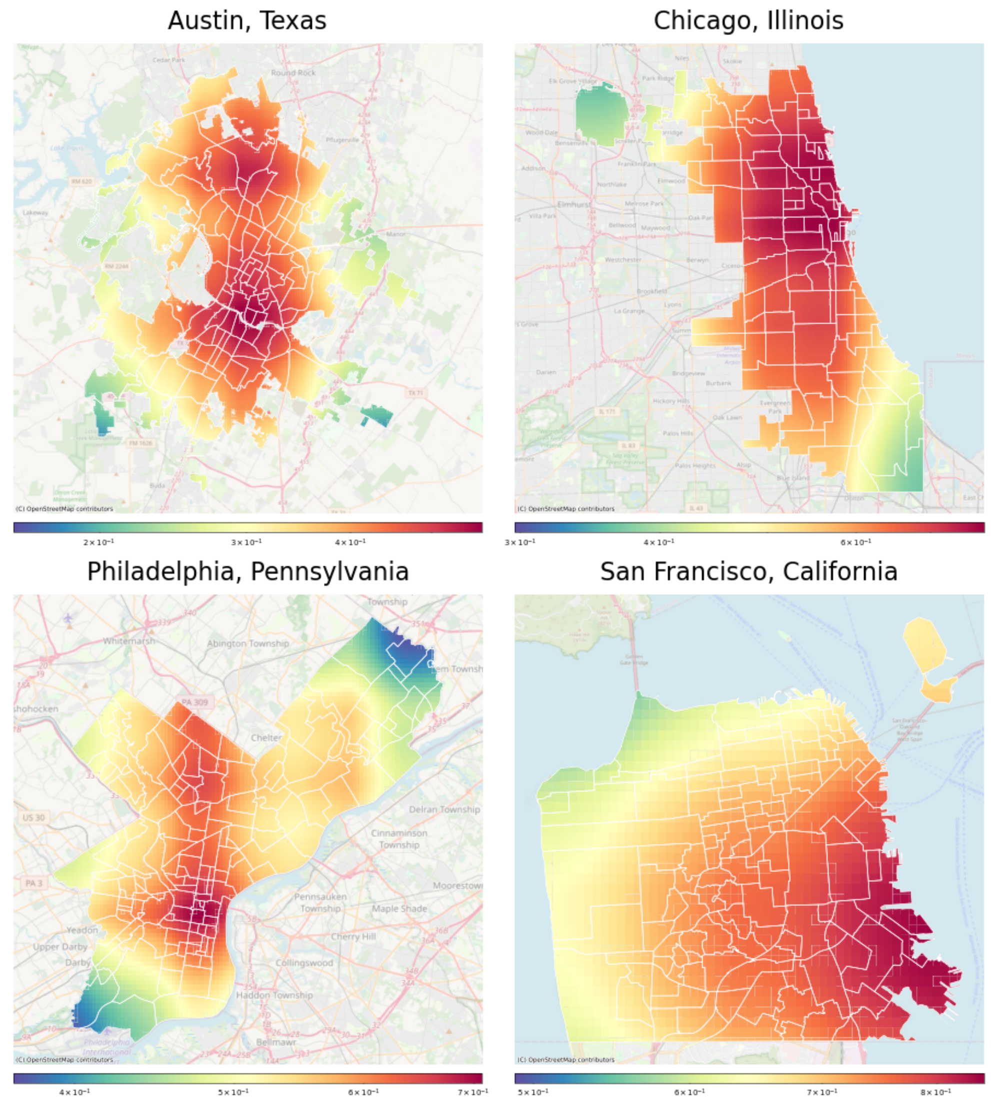

Figure 4.4: This image captures the weight values of the geospatial embedding matrix **H** defined in Section 3.2.3. It demonstrates the model's ability to identify patterns across different cities.

Conclusion

In conclusion, this thesis set out to make four contributions to the field of recommender systems while incorporating spatial information. We proposed a general-purpose location encoder architecture that is adaptable to any downstream task. We then applied this architecture to two specific tasks: **Category Probability Modeling** and **Rating Probability Distribution Modeling**. In these applications, we used users' locations together with their ratings to train models that recommend top categories based on geographic location.

To achieve these contributions, we began with a general introduction to the field of recommender systems, then we focused specifically on group recommendations and Point-of-Interest (POI) recommendations, which are an essential prerequisite to this work. We reviewed state-of-the-art recommendation algorithms that utilize concepts of groups and POIs as a foundation upon which we built our model. Additionally, we introduced the concept of location encoding, where we also reviewed the latest work and algorithms while outlining the properties a location encoder should possess to meet the requirements in our model.

We provided a detailed description of the datasets used in this thesis, including their processing and the reasoning behind each step. We also analyzed these datasets to prepare them for effective modeling.

Next, we proposed a novel location encoding model architecture that embeds geographic coordinates as separate latitudinal and longitudinal components. This architecture aligns with the location encoder properties outlined in Chapter 1. Based on this architecture, we developed two machine learning models to address the tasks of **Category Probability Modeling** and **Rating Probability Distribution Modeling**. These models aim to enable the creation of new POIs by recommending categories to groups of users at a common location.

In the final phase of our work, we set up experiments to validate the effectiveness of our proposed architecture. We trained the models on datasets from four different U.S. cities and evaluated them on previously unseen data. The results, compared against several baselines, demonstrated that our models provide high-quality recommendations, outperforming all baselines in terms of Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR).

Bibliography

1. TOBLER, Waldo R. A computer movie simulating urban growth in the Detroit region. *Economic geography*. 1970, vol. 46, no. sup1, pp. 234–240.

2. ADOMAVICIUS, G.; TUZHILIN, A. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*. 2005, vol. 17, no. 6, pp. 734–749. Available from DOI: 10.1109/TKDE.2005.99.

3. PAZZANI, Michael J.; BILLSUS, Daniel. Content-Based Recommendation Systems. In: *The Adaptive Web: Methods and Strategies of Web Personalization*. Ed. by BRUSILOVSKY, Peter; KOBSA, Alfred; NEJDL, Wolfgang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 325–341. ISBN 978-3-540-72079-9. Available from DOI: 10.1007/978-3-540-72079-9_10.

4. LESKOVEC, Jure; RAJARAMAN, Anand; ULLMAN, Jeffrey David. Recommendation Systems. In: *Mining of Massive Datasets*. 2nd. Cambridge University Press, 2014, chap. 9, pp. 319–353. ISBN 1107077230.

5. ADITYA, P. H.; BUDI, I.; MUNAJAT, Q. A comparative analysis of memory-based and model-based collaborative filtering on the implementation of recommender system for E-commerce in Indonesia: A case study PT X. In: *2016 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*. 2016, pp. 303–308. Available from DOI: 10.1109/ICACSIS.2016.7872755.

6. DARA, Sriharsha; CHOWDARY, C Ravindranath; KUMAR, Chintoo. A survey on group recommender systems. *Journal of Intelligent Information Systems*. 2020, vol. 54, no. 2, pp. 271–295.

7. BORATTO, Ludovico; CARTA, Salvatore. State-of-the-Art in Group Recommendation and New Approaches for Automatic Identification of Groups. In: *Information Retrieval and Mining in Distributed Environments*. Ed. by SORO, Alessandro; VARGIU, Eloisa; ARMANO, Giu-

liano; PADDEU, Gavino. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1–20. ISBN 978-3-642-16089-9. Available from DOI: `10.1007/978-3-642-16089-9_1`.

8. SHI, Jing; WU, Bin; LIN, Xiuqin. A Latent Group Model for Group Recommendation. In: *2015 IEEE International Conference on Mobile Services*. 2015, pp. 233–238. Available from DOI: `10.1109/MobServ.2015.41`.
9. FELFERNIG, Alexander; ATAS, Müslüm; HELIC, Denis; TRAN, Thi Ngoc Trang; STETTINGER, Martin; SAMER, Ralph. Algorithms for Group Recommendation. In: *Group Recommender Systems: An Introduction*. Ed. by FELFERNIG, Alexander; BORATTO, Ludovico; STETTINGER, Martin; TKALČIČ, Marko. Cham: Springer Nature Switzerland, 2024, pp. 29–61. ISBN 978-3-031-44943-7. Available from DOI: `10.1007/978-3-031-44943-7_2`.
10. PENNOCK, David M.; HORVITZ, Eric; LEE GILES, C. Social Choice Theory and Recommender Systems: Analysis of the Axiomatic Foundations of Collaborative Filtering. In: *Proceedings of the 17th National Conference on Artificial Intelligence and 12fth Conference on Innovative Applications ofArtificial Intelligence, AAAI 2000*. AAAI press, 2000, pp. 729–734. Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence, AAAI 2000. Funding Information: Thanks to Jack Breese and to the anonymous reviewers for ideas, insights, and pointers to relevant work. Publisher Copyright: Copyright © 2000, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.; 17th National Conference on Artificial Intelligence, AAA1 2000 ; Conference date: 30-07-2000 Through 03-08-2000.
11. SÁNCHEZ, Pablo; BELLOGÍN, Alejandro. Point-of-Interest Recommender Systems: A Survey from an Experimental Perspective. *CoRR*. 2021, vol. abs/2106.10069. Available from arXiv: `2106.10069`.
12. LEVANDOSKI, Justin J; SARWAT, Mohamed; ELDAWY, Ahmed; MOKBEL, Mohamed F. Lars: A location-aware recommender system. In: *2012 IEEE 28th international conference on data engineering*. IEEE, 2012, pp. 450–461.
13. LIAN, Defu; ZHAO, Cong; XIE, Xing; SUN, Guangzhong; CHEN, Enhong; RUI, Yong. GeoMF: joint geographical modeling and matrix factorization for point-of-interest recommendation. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, New York, USA: Association for Computing Machinery, 2014, pp. 831–840. KDD ’14. ISBN 9781450329569. Available from DOI: `10.1145/2623330.2623638`.

14. LI, Xutao; CONG, Gao; LI, Xiao-Li; PHAM, Tuan-Anh Nguyen; KRISHNASWAMY, Shonali. Rank-geofm: A ranking based geographical factorization method for point of interest recommendation. In: *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*. 2015, pp. 433–442.

15. YANG, Carl; BAI, Lanxiao; ZHANG, Chao; YUAN, Quan; HAN, Jiawei. Bridging Collaborative Filtering and Semi-Supervised Learning: A Neural Approach for POI Recommendation. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Halifax, NS, Canada: Association for Computing Machinery, 2017, pp. 1245–1254. KDD '17. ISBN 9781450348874. Available from DOI: `10.1145/3097983.3098094`.

16. MAI, Gengchen; JANOWICZ, Krzysztof; HU, Yingjie; GAO, Song; YAN, Bo; ZHU, Rui; CAI, Ling; LAO, Ni. A Review of Location Encoding for GeoAI: Methods and Applications. *CoRR*. 2021, vol. abs/2111.04006. Available from arXiv: `2111.04006`.

17. TANG, Kevin D.; PALURI, Manohar; FEI-FEI, Li; FERGUS, Rob; BOURDEV, Lubomir D. Improving Image Classification with Location Context. *CoRR*. 2015, vol. abs/1505.03873. Available from arXiv: `1505.03873`.

18. CHU, Grace; POTETZ, Brian; WANG, Weijun; HOWARD, Andrew; SONG, Yang; BRUCHER, Fernando; LEUNG, Thomas; ADAM, Hartwig. Geo-Aware Networks for Fine Grained Recognition. *CoRR*. 2019, vol. abs/1906.01737. Available from arXiv: `1906.01737`.

19. ZHANG, Tianyang; LI, Jiacheng. *Google Local Data Repository*. 2021. Available also from: `https://datarepo.eng.ucsd.edu/mcauley_group/gdrive/googlelocal/`. Accessed: 2023-11-03.

20. LI, Jiacheng; SHANG, Jingbo; MCAULEY, Julian. Uctopic: Unsupervised contrastive learning for phrase representations and topic mining. *arXiv preprint arXiv:2202.13469*. 2022. Available also from: `https://arxiv.org/abs/2202.13469`.

21. YAN, An; HE, Zhankui; LI, Jiacheng; ZHANG, Tianyang; MCAULEY, Julian. Personalized showcases: Generating multi-modal explanations for recommendations. In: *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2023, pp. 2251–2255. Available also from: `https://dl.acm.org/doi/abs/10.1145/3539618.3592036`.

22. *Philadelphia Neighborhoods* [`https://opendataphilly.org/datasets/philadelphia-neighborhoods/`]. [N.d.]. Accessed: 2023-11-03.

23. *Analysis Neighborhoods* [`https://data.sfgov.org/-/Analysis-Neighborhoods/p5b7-5n3h`]. [N.d.]. Accessed: 2023-11-03.

24. *Neighborhoods* [https://data.austintexas.gov/Locations-and-Maps/Neighborhoods/a7ap-j2yt/about_data]. [N.d.]. Accessed: 2023-11-03.

25. *Boundaries - Neighborhoods* [https://data.cityofchicago.org/Facilities-Geographic-Boundaries/Boundaries-Neighborhoods/bbvz-uum9]. [N.d.]. Accessed: 2023-11-03.

26. GEOPANDAS DEVELOPERS. *GeoPandas 0.11.1 Documentation*. GeoPandas, 2023. Available also from: https://geopandas.org/en/stable/. Accessed: 2023-11-1.

27. MALLIAROS, Fragkiskos D; GIATSIDIS, Christos; PAPADOPOULOS, Apostolos N; VAZIRGIANNIS, Michalis. The core decomposition of networks: Theory, algorithms and applications. *The VLDB Journal*. 2020, vol. 29, no. 1, pp. 61–92.

28. CHO, Eunjoon; MYERS, Seth A; LESKOVEC, Jure. Friendship and mobility: user movement in location-based social networks. In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2011, pp. 1082–1090.

29. LIU, Wei; LAI, Hanjiang; WANG, Jing; KE, Geyang; YANG, Weiwei; YIN, Jian. Mix geographical information into local collaborative ranking for POI recommendation. *World Wide Web*. 2020, vol. 23, pp. 131–152.

30. WIKIPEDIA CONTRIBUTORS. *Softmax function — Wikipedia, The Free Encyclopedia*. 2024. Available also from: https://en.wikipedia.org/w/index.php?title=Softmax_function&oldid=1220708012. [Online; accessed 9-May-2024].

31. FULMICOTON. *Bayesian Rating* [https://fulmicoton.com/posts/bayesian_rating/]. 2013. Accessed: 2024-03-12.

Appendix A

Contents of enclosed media

```
implementation ............................. directory with source code
trained-models.......................... directory with trained models
text ....................................... folder with the thesis text
  thesis.pdf................................... thesis in PDF format
  source...................... folder with the source code of the thesis
```