A

Major Project Report

On

POINT OF INTEREST RECOMMENDATION FOR LOCATION PROMOTION IN LOCATION BASED SOCIAL NETWORKS (LBSNs)

Submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

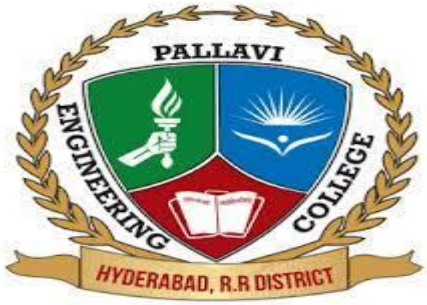
B. LALITHA PRIYA

186F1A0506

Under the guidance of

Mr. S. RAJU

M. Tech (CSE), Asst Prof



Department Of Computer Science and Engineering

PALLAVI ENGINEERING COLLEGE

(Approved by the AICTE, New Delhi & affiliated to JNTU, Hyderabad.)

Kuntloor (V), Hayathnagar (M), Hyderabad, R. R. Dist.-501 505.

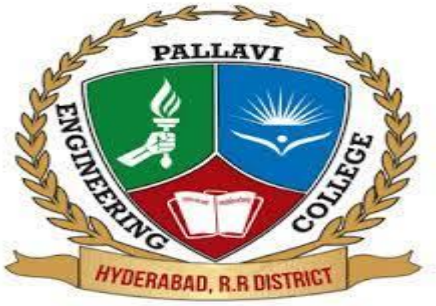
Telangana State

2022

Department Of Computer Science and Engineering

PALLAVI ENGINEERING COLLEGE

(Approved by the AICTE, New Delhi & affiliated to JNTU, Hyderabad.)
Kuntloor (V), Hayathnagar (M), Hyderabad, R. R. Dist.-501 505.
Telangana State, 2022



CERTIFICATE

This is to certify that the project entitled **"POINT OF INTEREST RECOMMENDATION FOR LOCATION PROMOTION IN LOCATION BASED SOCIAL NETWORKS"** is a bonafide work done by **B. LALITHA PRIYA (186F1A0506)** in partial fulfillment of the requirements for the award of the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING** by **JNTUH** during the Academic Year **2021-2022.**

The results presented in this thesis have been verified and are found to be satisfactory.The results embodied in this thesis have not been submitted to any other University for the award of any other degree or diploma.

INTERNAL GUIDE
Mr. S. RAJU
Asst Prof .Dept of CSE

HEAD OF THE DEPARTMENT
Mr. E. KRISHNA
BTech, MTech, Ph.D.

EXTERNAL VIVA-VOCE WAS HELD ON _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

I **Ms. B. LALITHA PRIYA (186F1A0506)** hereby declare that the project report entitled "**POINT OF INTEREST RECOMMENDATION FOR LOCATION PROMOTION IN LOCATION BASED SOCIAL NETWORKS**" is bonafide work done by me under the guidance of **Mr. S. RAJU, Asst. Prof., PALLAVI ENGINEERING COLLEGE**, during the year **2021-22**.This Project work is submitted to **JNTUH** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering. This work has not been submitted for any other institute for any degree, diploma previously.**

DATE: **B. LALITHA PRIYA**

PLACE: Kuntloor **186F1A0506**

ACKNOWLEDGEMENT

An endeavor of a long period can be successful only with the advice of my parents and well-wishers. I now take this opportunity to express my deep gratitude and appreciation to all those who encouraged me for successful completion of the project work.

I am very much thankful to **Sri M. Komaraiah, Hon'ble Chairman, Pallavi Group of Institutions** for his help in providing good facilities in our college.

I wish to express my sincere gratitude to **Dr. M. B. RAJU (B.Tech, MTech & Ph.D.), Professor & Principal**. His consistent help and encouragement to complete the project work.

My special thanks to **Mr. Erugu Krishna, (Assistant Prof. & Head of Dept, CSE), PALLAVI ENGINEERING COLLEGE** during the process of project work for his timely suggestions and help despite of his busy schedule.

I thank my guide **Mr. S. Raju**, **(Assoc Prof**, **Dept. of CSE), PALLAVI ENGINEERING COLLEGE** for his valuable guidance and suggestions in analyzing and testing throughout the period, till end of project work completion.

I also extend my thanks to the entire Teaching and Non-Teaching faculty of **PALLAVI ENGINEERING COLLEGE**, who have encouraged me throughout the course of **Bachelor's degree.**

Last but not least, I thank my family and all those helped me directly or indirectly for the completion of the project.

BY:

B. LALITHA PRIYA **186F1A0506**

ABSTRACT

Nowadays, Recommendation systems have become very popular. User can select an accurate item from large pool of items. Location characteristics play an important role in recommendation process. In recommendation systems, users' and items' current location, their previous location information and location histories of other users will impact decision of recommendation process. This paper conducts the study of various recommendation systems in which location characteristics are considered as key elements while recommendation. Recommendation techniques such as content based, collaborative and hybrid filtering, user preference model and matrix factorization methods are also studied. Location factor is considered as connective element between item and user in recommendation systems. With the wide application of location-based social networks (LBSNs), point-of-interest (POI) recommendation has become one of the major services in LBSNs. The behaviors of users in LBSNs are mainly checking in POIs, and these checking-in behaviors are influenced by user's behavior habits and his/her friends. In social networks, social influence is often used to help businesses to attract more users. Each target user has a different influence on different POI in social networks. This paper selects the list of POIs with the greatest influence for recommending users. Our goals are to satisfy the target user's service need, and simultaneously to promote businesses' locations (POIs). This paper defines a POI recommendation problem for location promotion. Additionally, we use sub-modular properties to solve the optimization problem. At last, this paper conducted a comprehensive performance evaluation for our method using two real LBSN datasets. Experimental results show that our proposed method achieves significantly superior POI recommendations comparing with other state-of-the-art recommendation approaches in terms of location promotion.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVATIONS

S.NO	Synonyms	Abbreviations
1	**LBSN**	Location Based Social Networks
2	**SVD**	Singular Vector Decomposition
3	**Sklearn**	Scikit-Learn
4	**POI**	Point of Interest
5	**RMSE**	Root Mean Square Error
6	**LBRS**	Location Based Recommendation Systems
7	**RS**	Recommendation Systems
8	**GPS**	Global Positioning System
9	**MSE**	Mean Square Error
10	**MF**	Matrix Factorization
11	**SQRT**	Square Root

CHAPTER - 1

INTRODUCTION

1.1. INTRODUCTION

In today's world of technology, various recommendation systems are available to users. Using those recommendation systems, users' time can be saved. Recommendation systems help the users to select suitable item from the large collection of items. It is difficult and tedious decision to select the proper one from huge collection of items. So, recommendation systems aid to recommend the appropriate results and save the selection time. In various recommendation systems, location characteristics play an important role. Nowadays, people use to tag their geo-locations, photos, and inform others about current location on various social media sites. Such location related information is useful in recommendation process. Location data helps to link the physical and digital domain and allow collecting the detail knowledge about particular user preferences and activities. Location is vital concept in recommendation systems which specify the relationship between users, between locations and between users and location [9]. Location helps to define the contextual information about any user. It is possible to collect detail knowledge about any item or user from their location histories. Location related information such as distance, weather, traffic conditions, address, timestamps, human density, reviews given by users for those locations, etc. are useful in recommendation systems. In recommendation systems, the current location of user, previous location information about users and location histories of different users affect the recommendation decisions. On the basis of those location related conditions recommendations are generated.

This paper conducts a study of various recommendation systems which are based on location factors. It includes restaurant recommendation, movie recommendation, outdoor recommendation, shopping recommendation and shop-type recommendation systems. There are various recommendation techniques used to generate recommendation results. This paper also focuses on study of various recommendation techniques and methods used in different recommendation systems. The emphasis of the research community on the problems of recommendation, such as cold start, data sparseness, and scalability emerged in the mid of 90s and RSs began to emerge as an independent research area. RSs have a profound association with

cognitive science (Bobrow 2014) and information retrieval (Lewis 2014). Different methods have been used for the recommendation problem and the methods are normally categorized as content-based and collaborative filtering methods.

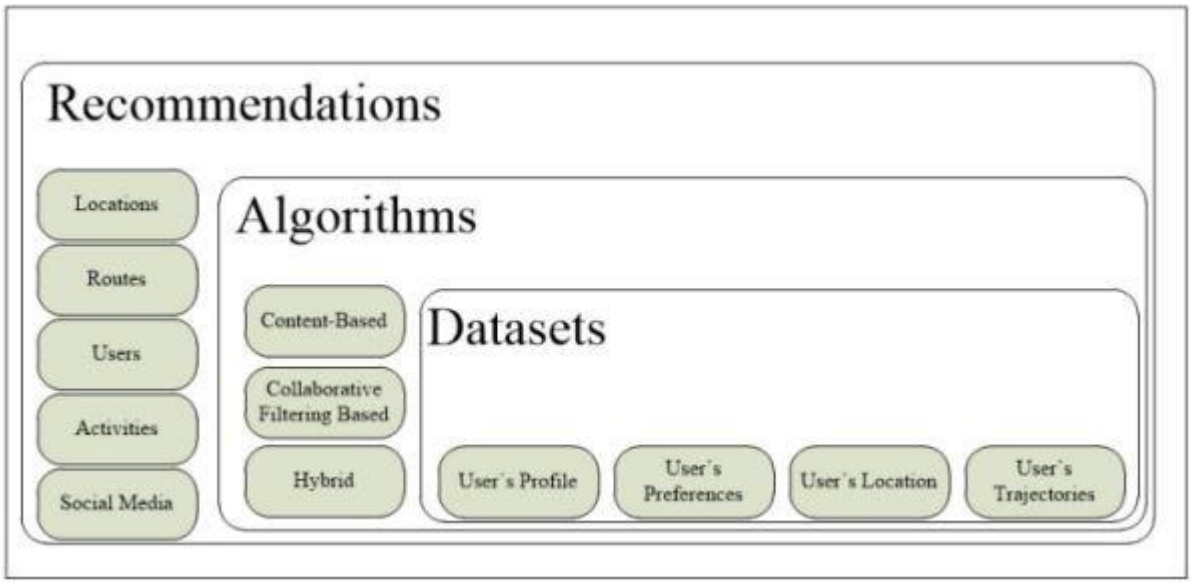


Figure 1.1 Components of Recommendation Systems

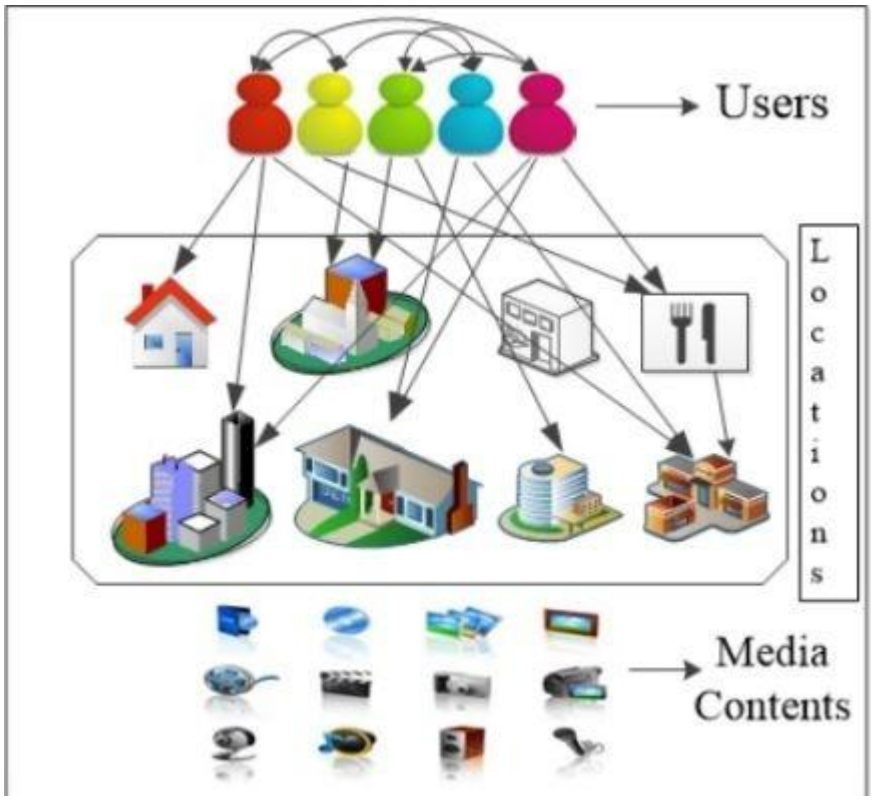


Figure 1.2 Visit patterns of Users in LBSNs

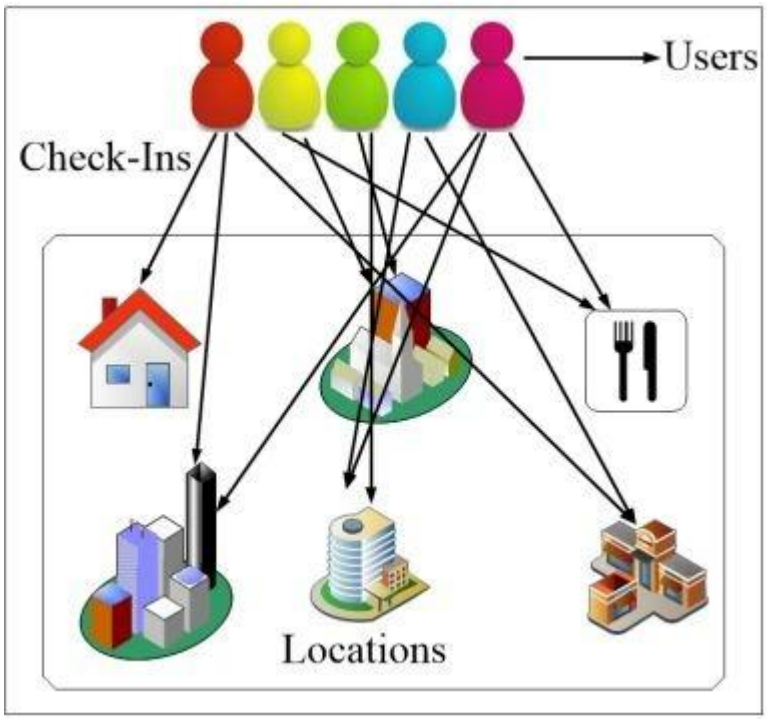


Figure 1.3 Basic Conflicts of Implicit Ratings using Check-ins

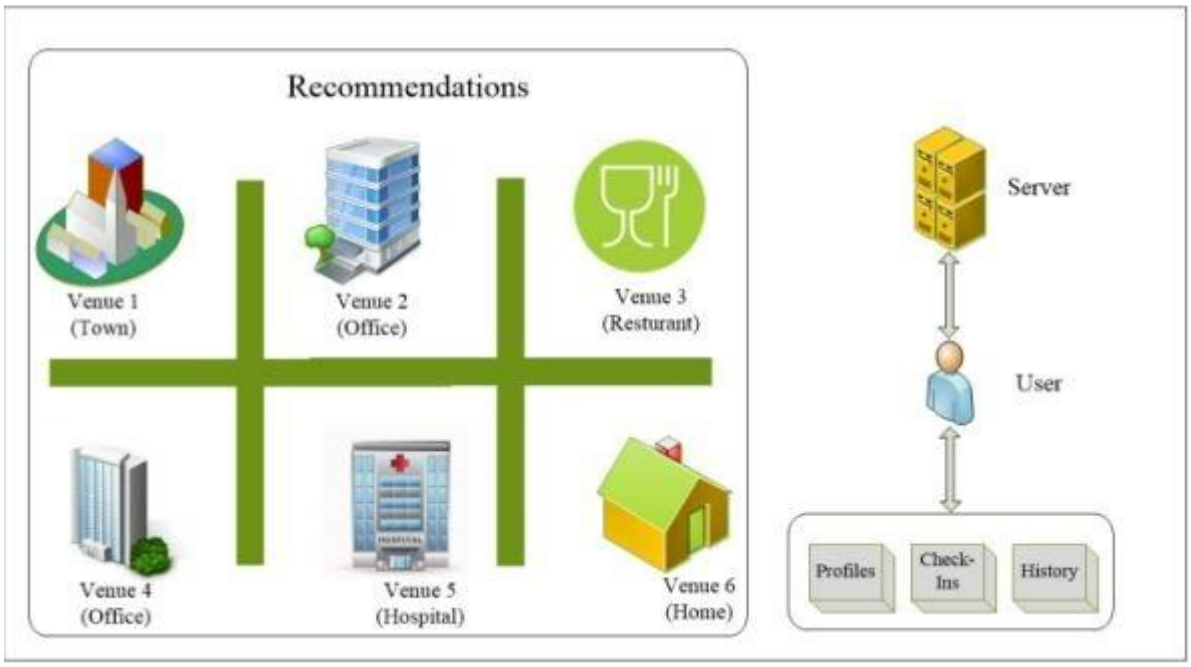


Figure 1.4 Overview of Location Recommendation

With the continuous evolution of social networking services, the significance of recommendation models considering group preference has also increased. However, most of the existing traditional recommendation schemes do not take into account group of friends' scenarios (Su et al. 2014); (Preot¸iuc-Pietro& Cohn 2013); (Noulas et al. 2011). In group scenarios, the RS not only models the preferences of a group member

but the location of each member must also be taken into account to satisfy all the members in the group. The individual's preferences and their preferred locations are then aggregated as the recommendation for the whole group (Christensen & Schiaffino 2014). There has been some limited work carried out in the recent past on group recommendations, such as (Hao et al. 2015); (Christensen & Schiaffino 2014); (Masthoff 2011). Most of the existing techniques proposed in the literature do not specifically focus on the effects of real-time physical factors, such as distance from location, traffic, and weather conditions on group recommendations. To achieve the objective of group recommendation, the following factors must be taken into account: (a) users' preferences, (b) current context (such as time and location), c) past check-ins, (d) geospatial characteristics, and (e) collaborative social opinion. However, the complexity and cost of processing the large-scale data sets negatively affect the performance and efficiency of RS. In the context of LBRS, there has been limited work performed on group-based location recommendations. In (Hao et al. 2015), the authors proposed a location-sensitive recommendations. The paper proposed an approach known as spatial social union which computes recommendations not only for a single user but also for group of users. The approach computes the similarity between two users and generates multiple matrices derived from user-user, user-item, and user-location graphs. In (Zhang et al. 2013), the authors proposed personalized event-based group recommendations. The main contribution of the work is group recommendations to the users living in the same city.

1.2 RECOMMENDER SYSTEMS

Recommendation engines are a subclass of machine learning which generally deal with ranking or rating products / users. Loosely defined, a recommender system is a system which predicts ratings a user might give to a specific item. These predictions will then be ranked and returned back to the user. They're used by various large name companies like Google, Instagram, Spotify, Amazon, Reddit, Netflix etc. often to increase engagement with users and the platform. For example, Spotify would recommend songs similar to the ones you've repeatedly listened to or liked so that you can continue using their platform to listen to music. Amazon uses recommendations to suggest products to various users based on the data they have collected for that user.

Recommender systems are often seen as a "black box", the model created by these large companies are not very easily interpretable. The results which are generated are often recommendations for the user for things that they need / want but are unaware that they need / want it until they've been recommended to them.

There are many different ways to build recommender systems, some use algorithmic and formulaic approaches like Page Rank while others use more modelling centric approaches like collaborative filtering, content based, link prediction, etc. All of these approaches can vary in complexity, but complexity does not translate to "good" performance. Often simple solutions and implementations yield the strongest results. For example, large companies like Reddit, Hacker News and Google have used simple formulaic implementations of recommendation engines to promote content on their platform.

1.2.1. RECOMMENDATION SYSTEM TYPES

1. Content Based (CB): The content-based approach is based on the contents, such as user profiles, attributes of the items etc. User profile shows user preferences or as another example based on the user profile, we can see which genre of movies he likes.

2. Collaborative Filtering (CF): The popular and most used approach can be divided into Memory-based and Model-based. The Memory-Based collaborative filtering approach can be divided into two main sections: User-Item filtering and Item-Item filtering.

- **User-User**

The most commonly used recommendation algorithm follows the *"people like you, like that"* logic. We call it a **"user-user"** algorithm because it recommends an item to a user if similar users liked this item before. The similarity between two users is computed from the number of items they have in common in the dataset.

- **Item-Item**

The **"item-item"** algorithm uses the same approach but reverses the view between users and items. It follows the logic *"if you like this, you might also like that"*. It recommends items that are similar to the ones you previously liked. As before the similarity between two items is computed using the number of users they have in common in the dataset.

Types of Collaborative Filtering :

i. Memory Based Collaborative Filtering:

As said before, Memory-Based Collaborative Filtering is divided into User-Item and Item-Item. The User-Item collaborative filtering method try to find similar users and based on the similar user's preferences (i.e., rating, checking etc.) recommend new things to users. In contrast, Item-Item filtering will take an item; find users who liked that item then find other items that those users or similar users also liked. So, based on the item taken it makes a recommendation.

The similarity values between items in Item-Item Collaborative Filtering are measured by observing all the users who have rated both items and for User-Item Collaborative Filtering, the similarity values between users are measured by observing all the locations that are checked-in (check-in frequency) by both users. There are some distance measures, such as Cosine Similarity, Jaccard Similarity etc. Cosine similarity is one the most used distance matrix in recommender systems. To this purpose, the check-in frequency considers as vectors in n-dimensional space and the similarity is calculated based on the angle between these vectors.

ii. Model Based Collaborative Filtering:

Model-Based CF uses machine learning and data mining approach to make a model to predict ratings. In recent years, most of the model-based approaches are based on Matrix Factorization (MF), and MF approaches received great attention because they are better than memory-based approaches for scalability and sparsity. As an unsupervised learning method, the goal of the MFs is to learn the latent preferences of users and latent attributes of items based on known ratings. Actually, in this method, a model tries to learn latent or hidden features of data characteristics to predict the unknown ratings. Consider a very sparse matrix with a lot of dimensions, MF restructures this matrix into a low-rank structure and we can represent it by multiplication of two low-rank matrices. The first matrix shows users and their latent preferences, as well as the second matrix shows the items and their latent attributes.

3. Hybrid Recommender Systems: Models that use both ratings and content features are called Hybrid Recommender Systems where both Collaborative Filtering and Content-based Models are combined.

CHAPTER - 2

LITERATURE SURVEY

With the increasing deployment and use of GPS-enabled devices, massive amounts of GPS data are becoming available. We propose a general framework for the mining of semantically meaningful, significant locations, e.g., shopping malls and restaurants, from such data. We present techniques capable of extracting semantic locations from GPS data. We capture the relationships between locations and between locations and users with a graph. Significance is then assigned to locations using random walks over the graph that propagates significance among the locations. In doing so, mutual reinforcement between location significance and user authority is exploited for determining significance, as are aspects such as the number of visits to a location, the durations of the visits, and the distances users travel to reach locations. Studies using up to 100 million GPS records from a confined spatio-temporal region demonstrate that the proposal is effective and is capable of outperforming baseline methods and an extension of an existing proposal. The problem of recommending tours to travelers is an important and broadly studied area. Suggested solutions include various approaches of points-of-interest (POI) recommendation and route planning. We consider the task of recommending a sequence of POIs that simultaneously uses information about POIs and routes. Our approach unifies the treatment of various sources of information by representing them as features in machine learning algorithms, enabling us to learn from past behavior. Information about POIs is used to learn a POI ranking model that accounts for the start and end points of tours. Data about previous trajectories are used for learning transition patterns between POIs that enable us to recommend probable routes. In addition, a probabilistic model is proposed to combine the results of POI ranking and the POI-to-POI transitions.

2.1. ARTICLES

1. Point-Of-Interest Recommender System for Social Groups.
AUTHORS: Ram Deepak Gottapu, Lakshmi VenkataSriram, Monangi

Used Supervised Learning Algorithm – KNN Algorithm and implemented signature-based group Recommender System using location based social networking data. The goal of this approach is to provide recommendations for groups of people with different sizes and relations. They introduced the concept of signatures for locations by mining the previously available user check-ins data. These signatures will provide recommendations to new groups.

The results demonstrate how the algorithm exploits the structure of the group and gives prediction based on the group rather than the approach of giving predictions to single user.

2. Point-of-interest lists and their potential in recommendation systems
AUTHORS: GiorgosStamatelatos, George Drosatos, SotiriosGyftopoulos, Helen Briola&Pavlos S. Efraimdis

They studied the potential of user generated Point-of-Interest lists in recommendation systems, proposed a methodology that only takes into consideration the bipartite graph structure of the POIs-lists graph, that is present in the Foursquare lists feature, to develop a POI recommendation system based on user profiles. This was made possible via similarity criteria that operate under this bipartite graph and can formulate the similarity between pairs of POIs. Their assumption was that information about the POIs is encoded in the lists and, in particular, that similar POIs will be submitted under the same list. Their evaluation confirmed, up to a certain extent that this assumption is reasonable. Specifically, they performed a user survey using local volunteers and used the responses to drive our algorithms and the evaluation process.

3. User Modeling for Point-of-Interest Recommendations in Location-Based Social Networks: The State of the Art

AUTHOR: Shudong Liu

This works by reviewing the taxonomy of user modeling for POI recommendations through the data analysis of LBSNs. First, they briefly introduced the structure and data characteristics of LBSNs, and then presented a formalization of user modeling for POI recommendations in LBSNs. Depending on which type of LBSNs data was fully utilized in user modeling approaches for POI recommendations, they divided user modeling algorithms into four categories: pure check-in data-based user modeling, geographical information-based user modeling, spatiotemporal information-based user modeling, and geo-social information-based user modeling. Finally, summarizing the existing works, they pointed out the future challenges and new directions in five possible aspects.

2.2 SYSTEM ANALYSIS

In this project, by using LBSNs we are going to explain the location recommendation based on the point of interest of the users with the help of the recommendation methods. We will mainly use the collaborative filtering by using the interaction between the user's and items.

In today’s world of technology, various recommendation systems are available to users. Using those recommendation systems, user’s time can be saved. Recommendation systems help the users to select suitable item from the large collection of items. In various recommendation systems, location characteristics play an important role. Nowadays, people use to tag their geo-locations, photos, and inform others about current location on various social media sites. Such location related information is useful in recommendation process.

2.2.1. Existing System

Recently, many researchers have been engaged in location-aware services. In LBSNs, users can post comments on locations or activities, upload photos, and share

check-in locations in which users are interested with friends. These locations are called points-of-interest (POIs). Currently, POI recommendation has become one of main location-aware services in LBSNs. POI recommendation approaches mostly involve recommending users with some locations in which users may be interested based on users' characters, preferences, and behavioral habits. Through the detailed analysis above, we observe traditional POI recommendations rarely focus on the effect of social relationships for businesses location promotion through the POI recommendation process.

2.2.2. Disadvantages of Existing System

There is no concept for the location promotion in LBSNs.
It helps only for business organizations not for users.

2.2.3. Proposed System

In view of POIs, POIs (e.g., restaurants, hotel, and markets) have to explore checking-in records to attract more users to visit; more users (e.g., friends of users that checked in these POIs) will be influenced to check in these locations. In this paper, we regard the influence on the business as a maximization location promotion problem. The essential goals of recommendation system are to satisfy users' service demands and merchants advertising needs. This paper proposes POI recommendation method for promoting POIs. Our proposed method is not only a tool for businesses to use to promote their products and attract more customers to visit their stores, but also recommends users with some POI's satisfying users' preferences.

2.2.4. Advantages of Proposed System

We propose a novel Point-of-Interest recommendation problem, and its goal is to promote the businesses' locations (POIs).
We define the user's IS under special POI categories in an entire social network, and model user mobility to describe the geographical influence between user.

CHAPTER - 3

REQUIREMENTS

3.1. INTRODUCTION

System requirements are the configuration that a system must have in order for a hardware or software application to run smoothly and efficiently. Failure to meet these requirements can result in installation problems or performance problems. The former may prevent a device or application from getting installed, whereas the latter may cause a product to malfunction or perform below expectation or even to hang or crash.

3.2. SOFTWARE REQUIREMENTS

For developing the application, the following are the Software Requirements:

1. Python Programming Language

Operating Systems supported

1. Windows 10 64 bit

Technologies and Languages used to Develop

1. Python 10
2. Google Colab

3.3. HARDWARE REQUIREMENTS

For developing the application, the following are the Hardware Requirements:

1. **Processor Type** : Intel i9
2. **Speed :** 3.40GHZ
3. **RAM :** 4GB DD2 RAM
4. **Hard Disk :** 500 GB
5. **Keyboard :** 101/102 Standard Keys
6. **Mouse :** Optical Mouse

CHAPTER - 4

SYSTEM DESIGN

4.1. INTRODUCTION

Recommender systems are more and more important in many big industries and some scales considerations have to be taken into account when designing the system (better use of sparsity, iterative methods for factorization or optimization, approximate techniques for nearest neighbor's search).

Recommender systems are difficult to evaluate: if some classical metrics such that MSE, accuracy, recall or precision can be used, one should keep in mind that some desired properties such as diversity (serendipity) and explain ability can't be assessed this way; real conditions evaluation (like A/B testing or sample testing) is finally the only real way to evaluate a new recommender system but requires a certain confidence in the model.

4.1.1. DATA FLOW DIAGRAM

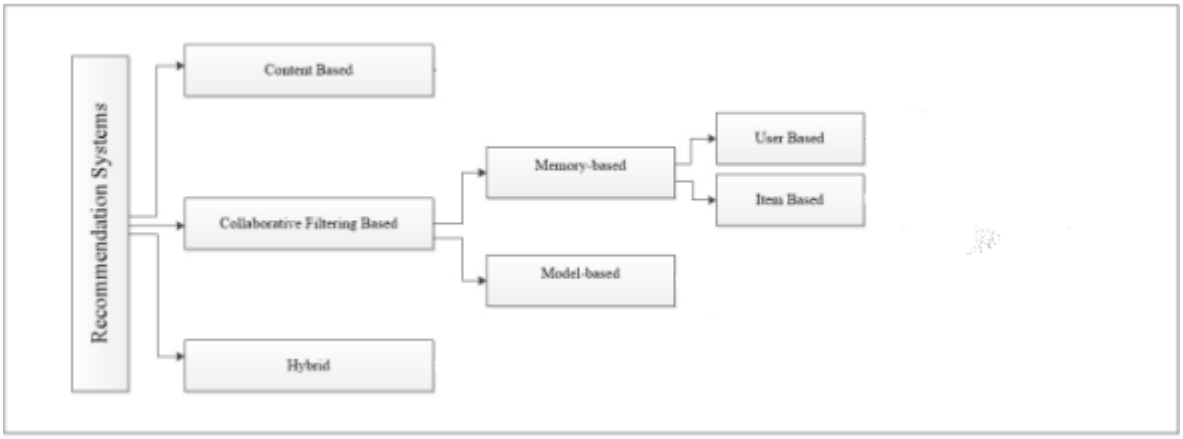


Fig 4.1 Flow Chart

4.2. SYSTEM ARCHITECTURE

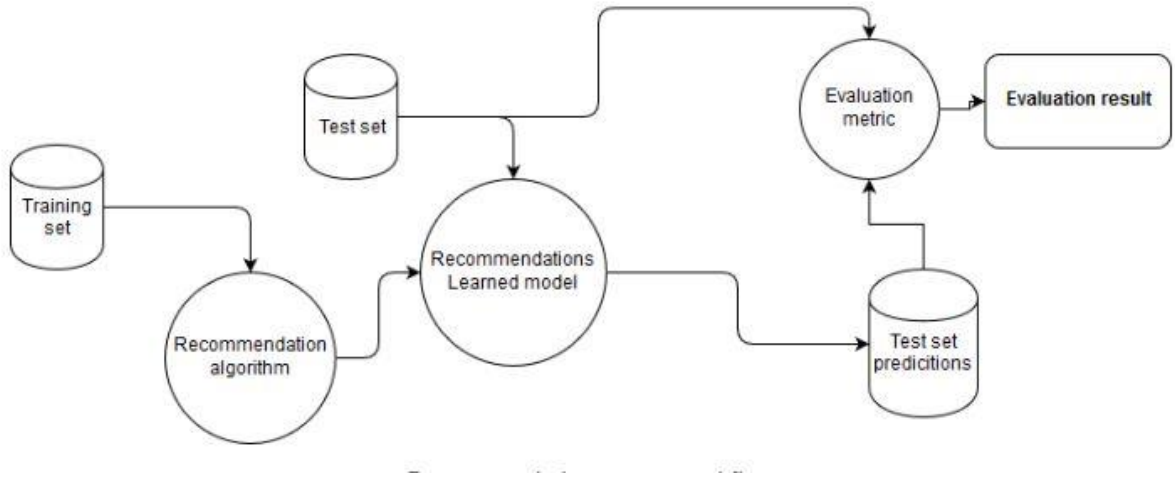


Figure 4.2 System Architecture

Training data set is a data set of examples used during the learning process and is used to fit the parameters (e.g., weights) of, for example, a classifier. For classification tasks, a supervised learning algorithm looks at the training data set to determine, or learn, the optimal combinations of variables that will generate a good predictive model. The goal is to produce a trained (fitted) model that generalizes well to new, unknown data. The fitted model is evaluated using "new" examples from the held-out datasets (validation and test datasets) to estimate the model's accuracy in classifying new data.

Test data set is a data set that is independent of the training data set, but that follows the same probability distribution as the training data set. If a model fit to the training data set also fits the test data set well, minimal overfitting has taken place. A better fitting of the training data set as opposed to the test data set usually points to over-fitting. A test set is therefore a set of examples used only to assess the performance (i.e., generalization) of a fully specified classifier. To do this, the final model is used to predict classifications of examples in the test set. Those predictions are compared to the examples' true classifications to assess the model's accuracy.

Recommender System: Each person has own preferences and personality. Recommender Systems try to use your preferences to personalize your experiences on the web and any other applications, and answer some questions like Who, Where, What etc. For example, where to eat? What to buy? Or even who you should be friends with?

Evaluation:

For evaluation there are many evaluation metrics (e.g., Root Mean Squared Error (RMSE), Precision, Recall etc.). The RMSE is one of the most popular evaluation metrics to evaluate predictions.

i. Precision

In the context of binary classification (Yes/No), precision measures the model's performance at classifying positive observations (i.e., "Yes"). In other words, when a positive value is predicted, how often is the prediction correct? We could game this metric by only returning positive for the single observation we are most confident in.

ii. Recall

Also called sensitivity. In the context of binary classification (Yes/No), recall measures how "sensitive" the classifier is at detecting positive instances. In other words, for all the true observations in our sample, how many did we "catch" We could game this metric by always classifying observations as positive.

iii. Recall vs. Precision

Say we are analyzing Brain scans and trying to predict whether a person has a tumor (True) or not (False). We feed it into our model and our model starts guessing.

Precision is the % of True guesses that were actually correct! If we guess 1 image is true out of 100 images and that image is actually true, then our precision is 100%! Our results aren't helpful however because we missed 10 brain tumors! We were super precise when we tried, but we didn't try hard enough.

Recall, or Sensitivity, provides another lens which with to view how good our model is. again, let's say there are 100 images, 10 with brain tumors, and we correctly guessed 1 had a brain tumor. Precision is 100%, but recall is 10%. Perfect recall requires that we catch all 10 tumors!

CHAPTER - 5

SOFTWARE SPECIFICATIONS

5.1. PYTHON

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. An interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java. It provides constructs that enable clear programming on both small and large scales. Python interpreters are available for many operating systems. C, Python, the reference implementation of Python, is open-source software and has a community-based development model, as do nearly all of its variant implementations. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3. The Python 2 language, i.e., Python 2.7.x, is "sun setting" on January 1, 2020 (after extension; first planned for 2015), and the Python team of volunteers will not fix security issues, or improve it in other ways after that date. With the end-of-life, only Python 3.5.x and later will be supported.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains C, Python, an open-source reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and C, Python development. Python strives for a simpler, less-cluttered syntax and grammar while giving developers a choice in their coding methodology.

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming.

Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

Why use Python?

Python is the programming language that is used to build the code in the right manner. The developers are using Python as it is a more coveted skillset after Swift. This programming language is in-demand, making companies hire Python developers for their projects. Here are the reasons to use Python.

1. Python is free of cost since it is an open-source programming language that is compatible with laptop and desktop. It supports modules, means, and libraries that are also free of cost.
2. Tech giants are also using Python programming language that makes it extremely popular.
3. Python is a programming language that can comprise the 7-8 line of code into just 2-3 lines with its easy syntax and requires less space.
4. Software developers can easily work on Python due to its series of definite coding and testing. It meets the actual results with the right input that includes the functionalities of the software.

Features of Python

As a programming language, the features of Python brought to the table are many. Some of the most significant features of Python are:

Figure 5.1 Features of Python

Easy to Code

Python is a very developer-friendly language which means that anyone and everyone can learn to code it in a couple of hours or days. As compared to other object-oriented programming languages like Java, C, C++, and C#, Python is one of the easiest to learn.

Open Source and Free

Python is an open-source programming language which means that anyone can create and contribute to its development. Python has an online forum where thousands of coders gather daily to improve this language further. Along with this Python is free to download and use in any operating system, be it Windows, Mac or Linux.

Support for GUI

GUI or Graphical User Interface is one of the key aspects of any programming language because it has the ability to add flair to code and make the results more visual. Python has support for a wide array of GUIs which can easily be imported to the interpreter, thus making this one of the most favorite languages for developers.

Object-Oriented Approach

One of the key aspects of Python is its object-oriented approach. This basically means that Python recognizes the concept of class and object encapsulation thus allowing programs to be efficient in the long run.

High-Level Language

Python has been designed to be a high-level programming language, which means that when you code in Python you don't need to be aware of the coding structure, architecture as well as memory management.

Integrated by Nature

Python is an integrated language by nature. This means that the python interpreter executes codes one line at a time. Unlike other object-oriented programming languages, we don't need to compile Python code thus making the debugging process much easier and efficient. Another advantage of this is, that upon execution the Python code is immediately converted into an intermediate form also known as byte-code.

Highly Portable

Suppose you are running Python on Windows and you need to shift the same to either a Mac or a Linux system, then you can easily achieve the same in Python without having to worry about changing the code. This is not possible in other programming languages, thus making Python one of the most portable languages available in the industry.

Highly Dynamic

As mentioned in an earlier paragraph, Python is one of the most dynamic languages available in the industry today. What this basically means is that the type of a variable is decided at the run time and not in advance. Due to the presence of this feature, we do not need to specify the type of the variable during coding, thus saving time and increasing efficiency.

Extensive Array of Library

Out of the box, Python comes inbuilt with a large number of libraries that can be imported at any instance and be used in a specific program. The presence of libraries also makes sure that you don't need to write all the code yourself and can import the same from those that already exist in the libraries.

Support for Other Languages

Being coded in C, Python by default supports the execution of code written in other programming languages such as Java, C, and C#, thus making it one of the versatile in the industry.

5.1.1. LIBRARIES

1. NumPy:

NumPy stands for numerical python. As the name gave it away, it's an open-source library for the Python programming language. NumPy is one of the most useful libraries.

Purpose of NumPy:

NumPy adds support for large, multi-dimensional matrices and arrays, along with a gigantic collection of top-end mathematical functions to operate on these arrays and matrices.

Its objective is to make it easier for you to transform difficult functions or calculate some data analysis. NumPy's biggest advantage is its fastness. It's so much faster than using the built-in Python's functions.

For example, it lets you simply calculate the mean and median of a data frame with a plain line of code for each:

How to install by using Anaconda Prompt

First off you need to install NumPy but only if you're not using Anaconda. To do so

- *Pip install numpy*
- *Conda install numpy*

How to import NumPy

- *import numpy*
- *import numpy as np*
- *from numpy import **
- *from numpy import add, subtract.*

2. PANDAS:

What is Pandas?

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Why use pandas?

Pandas allow us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science.

What Can Pandas Do?

Pandas gives you answers about the data.
Like:

- *is there a correlation between two or more columns?*
- *what is average value?*
- *Max value?*
- *Min value?*

Pandas are also able to delete rows that are not relevant, or contain wrong values, like empty or NULL values. This is called cleaning the data.

Installation of Pandas

If you have Python and PIP already installed on a system, then installation of Pandas is very easy.

Install it using this command:

C:\Users\Your Name>pip install pandas

If this command fails, then use a python distribution that already has Pandas installed like, *Anaconda,* and *Spyder* etc.

Import Pandas:

Once Pandas is installed, import it in your applications by adding the *import* keyword:
import pandas

Pandas as pd:

Pandas is usually imported under the *pd* alias.

import pandas as pd

3. MATPLOTLIB:

What is Matplotlib?

Matplotlib is a low-level graph plotting library in python that serves as a visualization utility. Matplotlib was created by John D. Hunter. Matplotlib is open source and we can use it freely. Matplotlib is mostly written in python, a few segments are written in C, Objective-C and JavaScript for Platform compatibility.

Installation of Matplotlib

If you have Python and PIP already installed on a system, then installation of Matplotlib is very easy.

Install it using this command:

C:\Users\Your Name>pip install matplotlib

Import Matplotlib

Once Matplotlib is installed, import it in your applications by adding the import module statement:

import matplotlib

Checking Matplotlib Version

The version string is stored under *__version__ attribute*

Example : *import matplotlib*

print (matplotlib __version__)

4. SKLEARN AND METRICS:

The Scikit-learn or sklearn library in Python. Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression and clustering and dimensionality reduction. Please note that sklearn is used to build machine learning models. It should not be used for reading the data, manipulating and summarizing it. There are better libraries for that (e.g., NumPy, Pandas etc.)

5.2 GOOGLE COLAB

Colab is a free notebook environment that runs entirely in the cloud. It lets you and your team members edit documents, the way you work with Google Docs. Colab supports many popular machine learning libraries which can be easily loaded in your notebook.

Google is quite aggressive in AI research. Over many years, Google developed AI framework called **TensorFlow** and a development tool called **Colaboratory**. Today TensorFlow is open-sourced and since 2017, Google made Colaboratory free for public use. Colaboratory is now known as **Google Colab** or simply **Colab**.

Another attractive feature that Google offers to the developers is the use of GPU. Colab supports GPU and it is totally free. The reasons for making it free for public could be to make its software a standard in the academics for teaching machine learning and data science. It may also have a long-term perspective of building a customer base for Google Cloud APIs which are sold per-use basis.

Irrespective of the reasons, the introduction of Colab has eased the learning and development of machine learning applications.

What Colab Offers You?

As a programmer, you can perform the following using Google Colab.

- Write and execute code in Python
- Document your code that supports mathematical equations

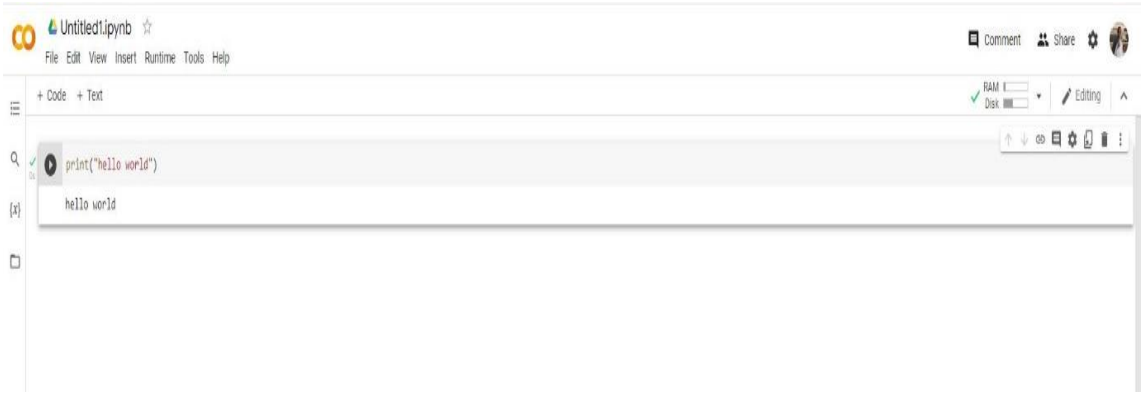
- Create/Upload/Share notebooks
- Import/Save notebooks from/to Google Drive
- Import/Publish notebooks from GitHub
- Import external datasets e.g., from Kaggle
- Integrate PyTorch, TensorFlow, Keras, OpenCV
- Free Cloud service with free GPU

How to get started

1. Go to https://colab.research.google.com/
2. Press **New notebook**, or if you already have a file, choose this one from the list
3. Save your notebook to either Google drive or GitHub via **File** > **Save a copy in …**

How to use Colab

Once you are in your Colab notebook you can use it just like any offline Jupyter notebook. Type your code in the darker gray box and press the run arrow to run the code. Your output will be placed below the box. To create a new code box, press **+ Code** in the navigation bar.



Import packages

Colab environment already has a number of pre-installed packages such as *pandas, PyTorch, SciPy, TensorFlow and Num*Py. To check all installed packages, use *pip freeze.*

```
!pip freeze
```

If a package is already installed, import these by importing them with *import {packages name}*

```
import pandas
```

If a package is not yet installed, install the package inside the Colab notebook by adding the *code!pip install {package name}*

```
!pip install cartopy
import cartopy
```

Adding collaborators:

To add other collaborators, go to the right corner and press *Share*. Now choose to either invite people via e-mail or by sending them a link. It is also possible to change the permissions to *'Commenter'* or '*Viewer'* by pressing  .

Connect to Google drive

If you want to import a dataset or any other file stored on your pc, the easiest way to do so is by connecting Google Colab to your Google drive. To do so, follow the steps below!

1. Upload your files to Google drive
2. Run the following lines in a Colab shell

```
from google.colab import drive
drive.mount('/content/drive')
```

3. Press **Connect to Google Drive**
4. Login to your Google account
5. Press **Allow**

CHAPTER - 6

IMPLEMENTATION

6.1. METHODOLOGIES

Machine Learning Classifiers are mainly divided into two categories: supervised learning and unsupervised learning. Supervised learning is also known as predictive learning that predicts the class of unknown objects based on prior class related information of similar objects. Unsupervised learning is also known as descriptive learning and finds patterns in unknown objects by grouping other similar objects together. According to the study, the Machine Learning Classifiers mainly used are as follows.

● **Supervised learning algorithms:** Think of any supervised machine learning algorithm you might have heard about and there is a very high chance that it is part of Scikit-learn. Starting from Generalized linear models (e.g., Linear Regression), Support Vector Machines (SVM), Decision Trees to Bayesian methods – all of them are part of Scikit-learn toolbox. The spread of machine learning algorithms is one of the big reasons for the high usage of Scikit-learn.

● **Cross-validation:** There are various methods to check the accuracy of supervised models on unseen data using *sklearn.*

● **Unsupervised learning algorithms:** Again, there is a large spread of machine learning algorithms in the offering – starting from clustering, factor analysis and principal component analysis to unsupervised neural networks.

● **Feature extraction:** Scikit-learn for extracting features from images and text (e.g., Bag of words).

There are 3 different APIs for evaluating the quality of a model's predictions:

● **Estimator score method:** Estimators have a score method providing a default evaluation criterion for the problem they are designed to solve. This is not discussed on this page, but in each estimator's documentation.

● **Scoring parameter:** Model-evaluation tools using cross-validation (such as *model_selection.cross_val_score* and *model_selection.GridSearchCV)* rely on an internal scoring strategy

● **Metric functions:** The *sklearn.Metrics* module implements functions assessing prediction error for specific purposes. These metrics are detailed in sections on Classification metrics, Multilabel ranking metrics, Regression metrics and Clustering metrics. Model selection and evaluation using tools, such as *model_selection.GridSearchCV* and *model_selection. cross_val_score,* take a scoring parameter that controls what metric they apply to the estimators evaluated. For the most common use cases, you can designate a scorer object with the scoring parameter; the table below shows all possible values. All scorer objects follow the convention that higher return values are better than lower return values. Thus, metrics which measure the distance between the model and the data, like *metrics.mean_squared_error,* are available as *neg_mean_squared_error* which return the negated value of the metric.

The module *sklearn.metrics* also exposes a set of simple functions measuring a prediction error given ground truth and prediction:

● Functions ending with _score return a value to maximize, the higher the better.

● Functions ending with *_error or _loss* return a value to minimize, the lower the better. When converting into a scorer object using *make_scorer,* set the greater_is_better parameter to False (True by default; see the parameter description below).

Metrics available for various machine learning tasks are detailed in sections below.

Many metrics are not given names to be used as scoring values, sometimes because they require additional parameters, such as *fbeta_score*. In such cases, you need to generate an appropriate scoring object. The simplest way to generate a callable object for scoring is by using *make_scorer*. That function converts metrics into callable that can be used for model evaluation.

The second use case is to build a completely custom scorer object from a simple python function using *make_scorer*, which can take several parameters:

- The python function you want to use (*my_custom_loss_func* in the example below)
- Whether the python function returns a score (*greater_is_better=True*, the default) or a loss (*greater_is_better=False*). If a loss, the output of the python function is negated by the scorer object, conforming to the cross-validation convention that scorers return higher values for better models.
- For classification metrics only: whether the python function you provided requires continuous decision certainties (*needs_threshold=True*). The default value is False.
- Any additional parameters, such as beta or labels in f1_score.

You can generate even more flexible model scorers by constructing your own scoring object from scratch, without using the *make_scorer* factory. For a callable to be a scorer, it needs to meet the protocol specified by the following two rules:

- It can be called with parameters (estimator, X, y), where estimator is the model that should be evaluated, X is validation data, and y is the ground truth target for X (in the supervised case) or none (in the unsupervised case).
- it returns a floating-point number that quantifies the estimator prediction quality on X, with reference to y. Again, by convention higher numbers are better, so if your scorer returns loss, that value should be negated.

Scikit-learn also permit evaluation of multiple metrics in *GridSearchCV*, *RandomizedSearchCV* and *cross_validate*.

The *sklearn.metrics* module implements several loss, score, and utility functions to measure classification performance. Some metrics might require probability estimates of the positive class, confidence values, or binary decisions values. Most implementations allow each sample to provide a weighted contribution to the overall score, through the *sample_weight* parameter.

The *accuracy_score* function computes the accuracy, either the fraction (default) or the count (normalize=False) of correct predictions.

In Multilabel classification, the function returns the subset accuracy. If the entire set of predicted labels for a sample strictly matches with the true set of labels, then the subset accuracy is 1.0; otherwise, it is 0.0.

- **Singular Value Decomposition (SVD):** SVD is basically a matrix factorization technique, which decomposes any matrix into 3 generic and familiar matrices. It has some cool applications in Machine Learning and Image Processing. To understand the concept of Singular Value Decomposition the knowledge on Eigen values and Eigen vectors is essential. A matrix factorization method generalizes the Eigen decomposition of a square matrix *(n x n)* to any matrix *(n x m)* (source).

You just need to know four things to understand the applications:

1. SVD is the decomposition of a matrix A into 3 matrices – *U, S,* and *V*
2. S is the diagonal matrix of singular values. Think of singular values as the importance values of different features in the matrix
3. The rank of a matrix is a measure of the unique information stored in a matrix. Higher the rank, more the information
4. Eigen vectors of a matrix are directions of maximum spread or variance of data.

In most of the applications, the basic principle of Dimensionality Reduction is used. You want to reduce a high-rank matrix to a low-rank matrix while preserving important information.

It minimizes the size of an image in bytes to an acceptable level of quality. This means that you are able to store more images in the same disk space as compared to before.

Image compression takes advantage of the fact that only a few of the singular values obtained after SVD are large. You can trim the three matrices based on the first few singular values and obtain a compressed approximation of the original image.

SVD for Spectral Clustering: Clustering is the task of grouping similar objects together. It is an unsupervised machine learning technique. For most of us, clustering is synonymous with K-Means Clustering – a simple but powerful algorithm. However, it is not always the most accurate.

- **Cosine Similarity:** Cosine similarity is one the most used distance matrix in recommender systems. To this purpose, the check-in frequency considers as vectors in n-dimensional space and the similarity is calculated based on the angle between these vectors. The **Cosine Similarity for user *u* and *u ′* is:**

$$sim(u,u') = cos(\theta) = \frac{r_u . \breve{r_{u'}}}{||r_u||\ ||r_{u'}||} = \sum_i \frac{r_{ui} . r_{u'i}}{\sqrt{\sum_i r_{ui}^2}\sqrt{\sum_i r_{u'i}^2}}$$

The similarity values between items in Item-Item Collaborative Filtering are measured by observing all the users who have rated both items and for User-Item Collaborative Filtering, the similarity values between users are measured by observing all the locations that are checked-in (check-in frequency) by both users. There are some distance measures, such as Cosine Similarity, Jaccard Similarity etc.

6.2. MODULES

i. Data Collection

It is the process of gathering and measuring information from countless different sources.

ii. Data Selection

Data selection is defined as the process of determining the appropriate data type and source, as well as suitable instruments to collect data. Data selection precedes the actual practice of data collection. We took data from online source named as Kaggle.

iii. Data Preprocessing

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data

preprocessing is a proven method of resolving such issues. Data preprocessing prepares raw data for further processing.

In this file, we will read the data and make some data preprocessing. First, we read the check-ins data file and calculate the check-in of each user and location pair with an indicator function (1 if a POI is checked by user u , otherwise 0).

iv. Data Classification

It is the process of analyzing structured or unstructured data and organizing it into categories based on file type, contents, and other metadata.

- **Training set**—a subset to train a model.
- **Testing set**—a subset to test the trained model.

We use the Scikit-learn library to split the dataset into testing and training.

model_selection.train_test_split shuffles and splits the data into two datasets according to the

percentage of test examples (*test_size*), which in this case is 0.20.

v. Collaborative Filtering:

In this filtering users and items (i.e., location) factors are used. It is further divided into two types,

1.Memory-Based Filtering.

2.Model-Based Filtering

vi. Evaluation Metrics:

Three methods are used: RMSE, Precision, Recall

➢ Root Mean Square Error (RMSE):

In machine learning, it is extremely helpful to have a single number to judge a model's performance, whether it is during training, cross-validation, or monitoring after deployment. **Root mean square error is one of the most widely used measures for this.**

It is a proper scoring rule that is intuitive to understand and compatible with some of the most common statistical assumptions.

By squaring errors and calculating a mean, RMSE can be heavily affected by a few

predictions which are much worse than the rest. If this is undesirable, using the absolute value of residuals and/or calculating median can give a better idea of how a model performs on most predictions, without extra influence from unusually poor predictions.

- **Precision:**

Precision *is* defined as the ratio of correctly classified positive samples (True Positive) to a total number of classified positive samples *(*either correctly or incorrectly*)*.

$$P = \frac{TruePositives}{TruePositives + FalsePositives}$$

- **Recall:**

The recall is calculated as the ratio between the numbers of Positive samples correctly classified as Positive to the total number of Positive samples. The **r**ecall measures the model's ability to detect positive samples. The higher the recall, the more positive samples detected.

$$R = \frac{TruePositives}{TruePositives + FalseNegatives}$$

6.3. DATASET INFORMATION

The original_data.csv file includes the check-ins information by users. Each line of the files follows the following format (tab separated format):

user_ID, POI_ID, coordinate(altitude and longitude), checkin_time (hour: in), date_id

p.s., check-ins made on the same date have the same ***date_id*** and there are **151589** check-ins

CHAPTER - 7

SNAPSHOTS

Recommender Systems in Location-Based Social Networks

```
[10] from google.colab import drive
     drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[11] import numpy as np
     import pandas as pd
     from sklearn import model_selection as ms
     from sklearn.metrics.pairwise import cosine_similarity
     from sklearn.metrics import mean_squared_error

     from math import sqrt
     import matplotlib.pyplot as plt
     from collections import defaultdict
```

```
[6]  header = ['user_id', 'location_id', 'frequency']
     df = pd.read_csv('/content/drive/MyDrive/DATASET/preprocessed_data.csv', sep='\t', names=header)

     n_users = df.user_id.unique().shape[0]
     n_locations = df.location_id.unique().shape[0]
     print('>> Number of users = ' + str(n_users))
     print('>> Number of locations = ' + str(n_locations))
```

```
>> Number of users = 2321
>> Number of locations = 5596
```

```
[8]  train_data, test_data = ms.train_test_split(df, test_size=0.20)
     print(">> Train data rows and columns:", train_data.shape)
     print(">> Test data rows and columns:", test_data.shape)
```

```
>> Train data rows and columns: (65942, 3)
>> Test data rows and columns: (16486, 3)
```

Memory based collaborative filtering

```
#Create two user-item matrices, one for training and another for testing
train_data_matrix = np.zeros((n_users, n_locations))

for checkin in train_data.itertuples():
    train_data_matrix[checkin[1], checkin[2]] = checkin[3]

# for RMSE
test_data_matrix = np.zeros((n_users, n_locations))
# for Precision and Recall
ground_truth_dic = defaultdict(set)

for checkin in test_data.itertuples():
    test_data_matrix[checkin[1], checkin[2]] = checkin[3]
    ground_truth_dic[int(checkin[1])].add(int(checkin[2]))
```

Cosine Similarity

```
[13] #user_similarity = cosine_similarity(train_data_matrix)
     #item_similarity = cosine_similarity(train_data_matrix.T)

     def cosine_similarity(train_matrix, kind='user', epsilon=1e-9):
         # epsilon -> small number for handling dived-by-zero errors
         if kind == 'user':
             sim = train_matrix.dot(train_matrix.T) + epsilon
         elif kind == 'location':
             sim = train_matrix.T.dot(train_matrix) + epsilon
         norms = np.array([np.sqrt(np.diagonal(sim))])
         return (sim / norms / norms.T)

     user_similarity = cosine_similarity(train_data_matrix, kind='user')
     item_similarity = cosine_similarity(train_data_matrix, kind='location')
```

```
def predict(checkins, similarity, type='user'):
    if type == 'user':
        pred = similarity.dot(checkins)/np.array([np.abs(similarity).sum(axis=1)]).T

    elif type == 'item':
        pred = checkins.dot(similarity)/np.array([np.abs(similarity).sum(axis=1)])

    return pred

user_prediction = predict(train_data_matrix, user_similarity, type='user')
item_prediction = predict(train_data_matrix, item_similarity, type='item')
```

Evaluation

```
[15] def rmse(prediction, ground_truth):
         # prediction[ground_truth.nonzero()] to only consider predicted checkins
         # (ratings) that are in the dataset.
         prediction = prediction[ground_truth.nonzero()].flatten()
         ground_truth = ground_truth[ground_truth.nonzero()].flatten()
         return sqrt(mean_squared_error(prediction, ground_truth))
```

```
UB_RMSE = rmse(user_prediction, test_data_matrix)
IB_RMSE = rmse(item_prediction, test_data_matrix)
print('>> User-based CF RMSE: ' + str(UB_RMSE))
print('>> Item-based CF RMSE: ' + str(IB_RMSE))
```

```
>> User-based CF RMSE: 0.967377830415953
>> Item-based CF RMSE: 0.979624473688774
```

Model Based Collaborative Filtering

```
[10] sparsity = round(1.0 - len(df) / float(n_users * n_locations), 3)
     print('>> The sparsity level is ' +  str(sparsity * 100) + '%')
```

```
>> The sparsity level is 99.4%
```

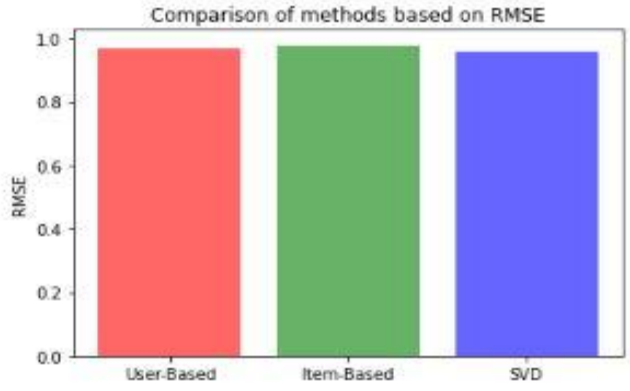
Singular Value Decomposition (SVD)

```
from scipy.sparse.linalg import svds

#get SVD components from train matrix. Choose k as r in formula.
u, s, vt = svds(train_data_matrix, k = 20)
s_diag_matrix = np.diag(s)
svd_prediction = np.dot(np.dot(u, s_diag_matrix), vt)
SVD_RMSE = rmse(svd_prediction, test_data_matrix)
print('>> Model-based CF RMSE: ' + str(SVD_RMSE))
```

```
>> Model-based CF RMSE: 0.9615272326894824
```

Experiments

```
objects = ('User-Based', 'Item-Based', 'SVD')
y_pos = np.arange(len(objects))
performance = [UB_RMSE,IB_RMSE,SVD_RMSE]

plt.bar(y_pos, performance, color=['red', 'green', 'blue'],
        align='center', alpha=0.6)
plt.xticks(y_pos, objects)
plt.ylabel('RMSE')
plt.title('Comparison of methods based on RMSE')
plt.show()
```

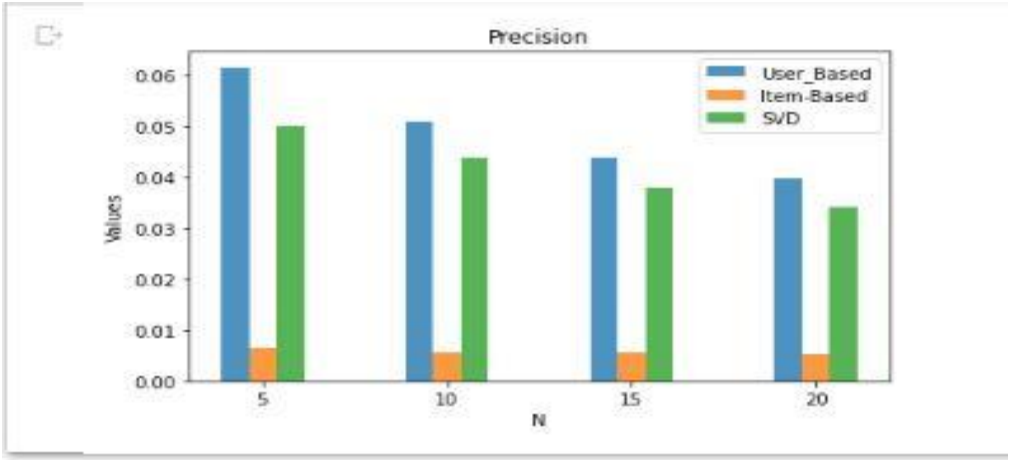


```
uPrecision5, uPrecision10, uPrecision15, uPrecision20 = [], [], [], []
iPrecision5, iPrecision10, iPrecision15, iPrecision20 = [], [], [], []
sPrecision5, sPrecision10, sPrecision15, sPrecision20 = [], [], [], []

uRecall5, uRecall10, uRecall15, uRecall20 = [], [], [], []
iRecall5, iRecall10, iRecall15, iRecall20 = [], [], [], []
sRecall5, sRecall10, sRecall15, sRecall20 = [], [], [], []

def precision_k(actual, predicted):
    return 1.0 * len(set(actual) & set(predicted)) / len(predicted)


def recall_k(actual, predicted):
    return 1.0 * len(set(actual) & set(predicted)) / len(actual)

all_uids = list(range(n_users))
all_lids = list(range(n_locations))
np.random.shuffle(all_uids)

for cnt, uid in enumerate(all_uids):
    if uid in ground_truth_dic:
        user_scores = [user_prediction[uid, lid]
                       if train_data_matrix[uid, lid] == 0 else -1
                           for lid in all_lids]
```

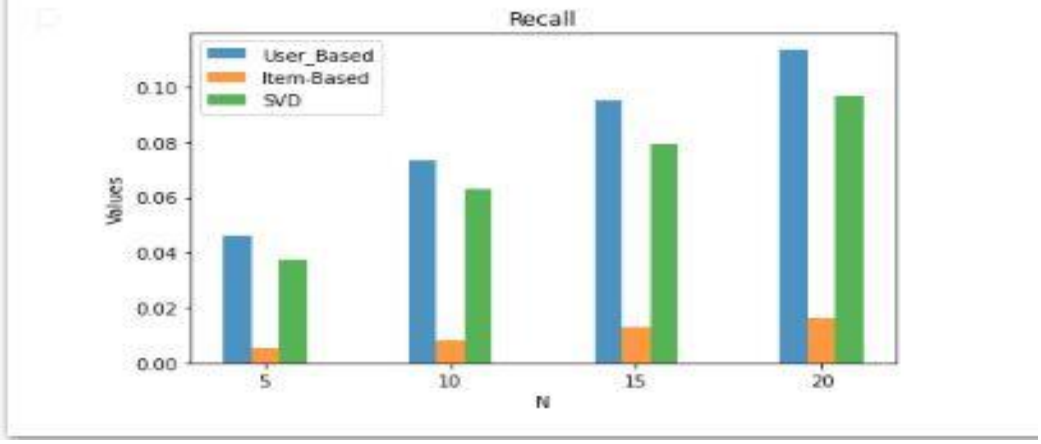
```
item_scores = [item_prediction[uid, lid]
                if train_data_matrix[uid, lid] == 0 else -1
                    for lid in all_lids]

svd_scores = [svd_prediction[uid, lid]
                if train_data_matrix[uid, lid] == 0 else -1
                    for lid in all_lids]

user_scores = np.array(user_scores)
item_scores = np.array(item_scores)
svd_scores = np.array(svd_scores)

user_predicted = list(reversed(user_scores.argsort()))[:100]
item_predicted = list(reversed(item_scores.argsort()))[:100]
svd_predicted = list(reversed(svd_scores.argsort()))[:100]

actual = ground_truth_dic[uid]

k = [5, 10, 15, 20]

for i in k:
    u_precision = precision_k(actual, user_predicted[:i])
    u_recall = recall_k(actual, user_predicted[:i])

    i_precision = precision_k(actual, item_predicted[:i])
    i_recall = recall_k(actual, item_predicted[:i])

    s_precision = precision_k(actual, svd_predicted[:i])
    s_recall = recall_k(actual, svd_predicted[:i])
```

```
    if i == 5:
        uPrecision5.append(u_precision)
        iPrecision5.append(i_precision)
        sPrecision5.append(s_precision)

        uRecall5.append(u_recall)
        iRecall5.append(i_recall)
        sRecall5.append(s_recall)

    elif i == 10:
        uPrecision10.append(u_precision)
        iPrecision10.append(i_precision)
        sPrecision10.append(s_precision)

        uRecall10.append(u_recall)
        iRecall10.append(i_recall)
        sRecall10.append(s_recall)

    elif i == 15:
        uPrecision15.append(u_precision)
        iPrecision15.append(i_precision)
        sPrecision15.append(s_precision)

        uRecall15.append(u_recall)
        iRecall15.append(i_recall)
        sRecall15.append(s_recall)

    elif i == 20:
        uPrecision20.append(u_precision)
        iPrecision20.append(i_precision)
        sPrecision20.append(s_precision)

        uRecall20.append(u_recall)
        iRecall20.append(i_recall)
        sRecall20.append(s_recall)
```

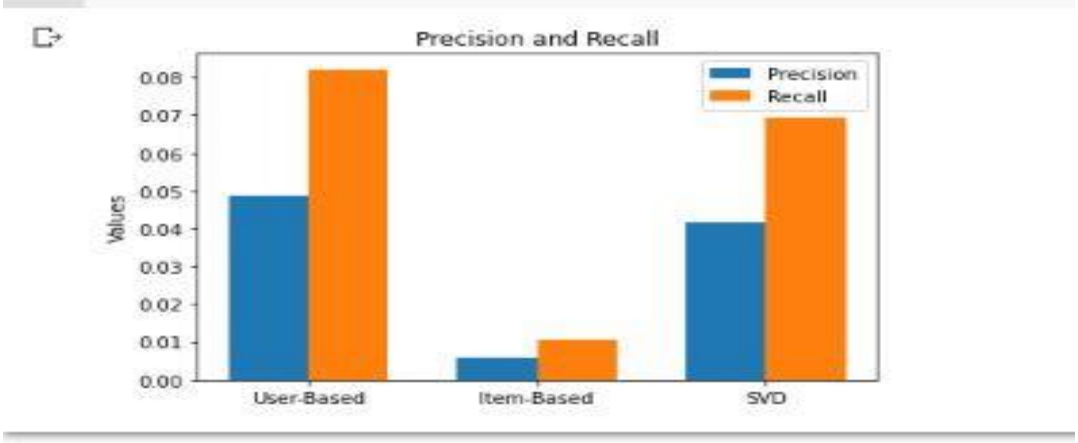
```
# data to plot
n_groups = 4
user = (np.mean(uPrecision5), np.mean(uPrecision10),
        np.mean(uPrecision15), np.mean(uPrecision20))
item = (np.mean(iPrecision5), np.mean(iPrecision10),
        np.mean(iPrecision15), np.mean(iPrecision20))
svd = (np.mean(sPrecision5), np.mean(sPrecision10),
       np.mean(sPrecision15), np.mean(sPrecision20))

# create plot
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.15
opacity = 0.8

rects1 = plt.bar(index, user, bar_width, alpha=opacity, label='User_Based')
rects2 = plt.bar(index + bar_width, item, bar_width,
                 alpha=opacity, label='Item-Based')
rects3 = plt.bar(index + bar_width + bar_width, svd,
                 bar_width, alpha=opacity, label='SVD')

plt.xlabel('N')
plt.ylabel('Values')
plt.title('Precision')
plt.xticks(index + bar_width, ('5', '10', '15', '20'))
plt.legend()

plt.tight_layout()
plt.show()
```




```
# data to plot
n_groups = 4
user = (np.mean(uRecall5), np.mean(uRecall10),
        np.mean(uRecall15), np.mean(uRecall20))
item = (np.mean(iRecall5), np.mean(iRecall10),
        np.mean(iRecall15), np.mean(iRecall20))
svd = (np.mean(sRecall5), np.mean(sRecall10),
       np.mean(sRecall15), np.mean(sRecall20))

# create plot
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.15
opacity = 0.8

rects1 = plt.bar(index, user, bar_width, alpha=opacity, label='User_Based')
rects2 = plt.bar(index + bar_width, item, bar_width,
                 alpha=opacity, label='Item-Based')
rects3 = plt.bar(index + bar_width + bar_width, svd,
                 bar_width, alpha=opacity, label='SVD')

plt.xlabel('N')
plt.ylabel('Values')
plt.title('Recall')
plt.xticks(index + bar_width, ('5', '10', '15', '20'))
plt.legend()

plt.tight_layout()
plt.show()
```



```
N = 3
means_precision = (np.mean(uPrecision5 + uPrecision10 + uPrecision15 + uPrecision20),
                   np.mean(iPrecision5 + iPrecision10 + iPrecision15 + iPrecision20),
                   np.mean(sPrecision5 + sPrecision10 + sPrecision15 + sPrecision20))

means_recall = (np.mean(uRecall5 + uRecall10 + uRecall15 + uRecall20),
                   np.mean(iRecall5 + iRecall10 + iRecall15 + iRecall20),
                   np.mean(sRecall5 + sRecall10 + sRecall15 + sRecall20))

ind = np.arange(N)
width = 0.35

plt.bar(ind, means_precision, width, label='Precision')
plt.bar(ind + width, means_recall, width, label='Recall')

plt.ylabel('Values')
plt.title('Precision and Recall')

plt.xticks(ind + width / 2, ('User-Based', 'Item-Based', 'SVD'))
plt.legend(loc='best')
plt.show()
```



CHAPTER - 8

APPLICATIONS

8.1. INTRODUCTION

It is easy to get confused about recommendation systems as they are also called recommender systems or recommendation engines. All of these perform the same actions; they are systems that predict what your customers want by analyzing their behavior which contains information on past preferences. This chapter gives the applications of the same.

8.2. APPLICATIONS

i. Healthcare

Healthcare is one of the main areas where LBRS can significantly enhance the efficiency, reliability and effectiveness of the system (Wiesner & Pfeifer 2014); (Hoens et al. 2013) ;(Middleton et al.2013). People from various domains often require multiple healthcare services, such as disease-specific specialist, hospitals, and health insurance plans (Abbas et al. 2015) that closely matcha user's preferences. LBRS can play an important role in the healthcare industry in order to connect and provide localized recommendations for patients, healthcare providers, and insurance companies.

ii. Transportation

Another interesting area for the adoption of LBRS is transportation. LBRS can be helpful in route recommendations, e.g., for individuals driving their personal vehicles, cab drivers, and public transporters (Liu et al. 2014); (Su et al. 2014). Heavy traffic in peak hours is one of the significant problems all over the world. In such situations, people can use the services of LBRS for different routes to their destinations. Similarly, carpooling is also among one of the services provided by LBRS (Zhang et al. 2014). Effective LBRS adoptions in transportation can significantly reduce the cost of fuel and enhance the reliability of users in the services provided by LBRS.

iii. Tourism

An important area where LBRS are actively deployed is the tourism industry where people want to plan in advance their preferred locations to visit. Sometimes it is difficult to choose the appropriate place when you have to make a decision from multiple available choices. LBRS has been used to provide the effectiveness in tourism by recommending the appropriate trip plans, as well as the other well-known nearby point-of-interests like hotels, restaurants, shopping malls(Le &Pishva 2016); (Chen et al. 2013). The adoption of LBRS in tourism can significantly save the time of tourists to reach their destination in a suitable time.

iv. Education

Another key area where the services of LBRS can significantly play an important role is education. Students need better institutions such as colleges and universities for their higher education. LBRS can be used to discover the best institutes according to the preferences of the student.

CHAPTER - 9

CONCLUSION AND FUTURE ENHANCEMENT

9.1. CONCLUSION

Now, we have implemented two basic approaches for location-based recommender systems. As you saw, the Memory-based CF algorithms are easy to implement and the result has appropriate prediction quality. The main challenges in Memory-based approaches are scalability and cold-start problems. They are not suitable for a real-world scenario, and they have problems when a new user or item is added to the system (i.e., cold-start). In contrast, Model-Based CF approaches are scalable and can handle when data is sparse but also suffer from the cold-start problem. Also, more recent work in the Model-based CF minimizes the squared error by applying alternating least square or stochastic gradient descent and uses regularization terms to prevent over fitting. In various recommendation systems, location information about items and users helps to generate the recommendation results. Location data helps to link the physical and digital domain and allows collecting detailed knowledge about particular user preferences and activities. Location is a vital concept in recommendation systems which specify the relationship between users, between locations and between users and location. Various recommendation systems use location factor as a major component to find out the correct and accurate results. Location information is considered as a context in various content based and contextual recommendation techniques. Location information is useful to calculate the accurate and effective recommendations in different recommendation systems.

9.2. FUTURE ENHANCEMENT

To improve the efficiency of recommendations offered by LBRS, some important factors need to be considered. Such factors include:

(a) When the system is offering routes along with locations, then the key location and routes

attributes like location type, distance of the location, travel time, route complexity, time-of-day, and real-time world factors with temporal features need to be considered.

(b) For better results and customization point of view, users' customization to plan their visits
using platforms like smart phones need to be considered.

(c) Most of the existing LBRS use only a single type of data source for recommendations.
Using diverse data sources will enable the LBRS to provide effective recommendations. Diverse data sources may include distance from location, traffic conditions, weather conditions, multiple routes to location, and time of the day (morning, evening or night).

(d) For achieving the best efficiency of recommendations, hybrid techniques need to be considered that will overcome the limitations of the existing techniques.

(e) Real-time factors and group features are also ignored in most of the existing literature. The main motivation behind consideration of real-world parameters is to include the current context of each of the group member in the location recommendation process. By doing so, the selected location will be based on mutual consensus of group members and will be the one that satisfies all the members in a group. It is noteworthy to mention that providing real-time recommendations is highly compute-intensive task as the workload consists of huge volumes of users' data accumulated in the system over time.

CHAPTER - 10

REFERENCES & BIBLIOGRAPHY

[1] Jun Zeng, Feng Li, Haiyang Liu, Junhao Wen, Sachio Hirokava, "A Restaurant Recommender System Based on User Preference and Location in Mobile Environment," in 5th IIAI International Congress on Advanced Applied Informatics, 2016.

[2] KasraMadadipouya, "A Location based Movie Recommender System Using Collaborative Filtering," in International Journal in Foundations of Computer Science & Technology (IJFCST), Vol.5, No.4, July 2015.

[3] Yuichiro Takeuchi, Masanori Sugimoto, "An Outdoor Recommendation System based on User Location History," in ubiPCMM, 2005.

[4] Wan-Shiou Yang, Hung-Chi Cheng, Jia-Ben Dia, "A location-aware recommender system for mobile shopping environments," in Expert Systems with Applications, 437–445, Elsevier, 2008.

[5] Zhiwen Yu, Miao Tian, Zhu Wang, And Bin Guo, "ShopType Recommendation Leveraging the Data from Social Media and Location-Based Services," in ACM Transactions on Knowledge Discovery from Data, Vol. 11, No. 1, Article 1, July 2016.

[6] Hongzhi Yin, Bin Cui, Yizhou Sun, Zhiting Hu, Ling Chen, "LCARS: A Spatial Item Recommender System," in ACM Trans. Inf. Syst. V, N, Article A, 35 pages, 2014.

[7] https://en.wikipedia.org/wiki/Recommender_system

[8] Tomas Horvath, "A Model of User Preference Learning for Content-based Recommender Systems," in Computing and Informatics, Vol. 29, 1001–1029, 2008.

[9] Jie Bao, Yu Zheng, David Wilkie, and Mohamed Mokbel "Recommendations in location-based social networks: A survey," in GeoInformatica, 525565, 2015

[10] https://www.sciencedirect.com/science/article/pii/S1877050917318148

[11] https://link.springer.com/article/10.1007/s40558-021-00195-5

[12] https://www.hindawi.com/journals/misy/2018/7807461/

[13] https://www.researchgate.net/publication/312480566_A_comparative_study_of_location-based_recommendation_systems

[14] https://en.wikipedia.org/wiki/Training,_validation,_and_test_data_sets

[15] https://ml-cheatsheet.readthedocs.io/en/latest/glossary.html

Point of Interest Recommendation for Location Promotion in Location-based Social Networks

Mr. Sabavath Raju[1]
Assistant Professor, CSE,
srajunayak@gmail.com
Pallavi Engineering College, Hyderabad

Ms. B. Lalitha Priya[2]
186F1A0506, UG Scholar, CSE,
Lpriya9307@gmail.com
Pallavi Engineering College, Hyderabad

Mr. Golla Govardhan[3]
186F1A0517, UG Scholar, CSE,
govardhangolla012@gmail.com
Pallavi Engineering College, Hyderabad

Mr. Alok Chand[4]
186F1A0502, UG Scholar, CSE,
Alokchand2000@gmail.com
Pallavi Engineering College, Hyderabad

Abstract
With the wide application of location-based social networks (LBSNs), point-of-interest (POI) recommendation has become one of the major services in LBSNs. The behaviors of users in LBSNs are mainly checking in POIs, and these checking-in behaviors are influenced by user's behavior habits and his/her friends. In social networks, social influence is often used to help businesses to attract more users. Each target user has a different influence on different POI in social networks. This paper selects the list of POIs with the greatest influence for recommending users. Our goals are to satisfy the target user's service need, and simultaneously to promote businesses' locations (POIs). This paper defines a POI recommendation problem for location promotion. Additionally, we use sub-modular properties to solve the optimization problem. At last, this paper conducted a comprehensive performance evaluation for our method using two real LBSN datasets. Experimental results show that our proposed method achieves significantly superior POI recommendations comparing with other state-of-the-art recommendation approaches in terms of location promotion.

Keywords: Location-Based Social Networks, Recommender systems, Location Promotion

1. INTRODUCTION

In today's world of technology, various recommendation systems are available to users. Using those recommendation systems, user's time can be saved. Recommendation systems help the users to select suitable item from the large collection of items. It is difficult and tedious decision to select the proper one from huge collection of items. So, recommendation systems aid to recommend the appropriate results and save the selection time. In various recommendation systems, location characteristics play an important role. Nowadays, people use to tag their geo-locations, photos, and inform others about current location on various social media sites. Such location related information is useful in recommendation process. Location data helps to link the physical and digital domain and allows collecting the

detailed knowledge about particular user preferences and activities. Location is vital concept in recommendation systems that specify the relationship between users, between locations and between users and location [9]. Location helps to define the contextual information about any user. Such contextual information is used to generate recommendations results. It is possible to collect detail knowledge about any item or user from their location histories. Location related information such as distance, weather, traffic conditions, address, timestamps, human density, reviews given by users for those locations, etc. are useful in recommendation systems. In recommendation systems, the current location of user, previous location information about users and location histories of different users affect the recommendation decisions. On the basis of those location related conditions recommendations are generated. This paper conducts a study of various recommendation systems which are based on location factors. It includes restaurant recommendation, movie recommendation, outdoor recommendation, shopping recommendation and shop-type recommendation systems. There are various recommendation techniques used to generate recommendation results. This paper also focuses on study of various recommendation techniques and methods used in different recommendation systems.

2. Problem formulations

In this paper, we focus on the problem of POI recommendation by mining their historical behavior data in LBSNs. For ease of presentation, we define the key data structures and notations used in this paper, which are summarized in Table 1.1. Definition 1 (POI) A POI is a geographical point with specific functions (e.g., hotel, restaurant, museum, store) that user may find useful or interesting

Variable	Interpretation
U, R, V	the set of users, locations and POIs
W	the vocabulary set
D_u	the profile of user u
$v_{u,i}$	the POI of i^{th} record in D_u
$l_{u,i}$	the location of POI $v_{u,i}$
l_u	the home location of the user u
$W_{u,i}$	the set of words describing POI $v_{u,i}$
$w_{u,i,n}$	the n^{th} content word describing POI $v_{u,i}$
$t_{u,i}$	the time of i^{th} check-in record in D_u

TABLE 1.1: Notations of The Input Data

In our model, POI has three attributes: identifier, location and content. We use v to represent a POI identifier, lv to denote its corresponding location identifier and Wv to represent the set of words describing the POI (e.g., tags and categories). The location information available for each POI v in the collected raw datasets is in the form of the (latitude, longitude) pair. Then, index structures are applied to partition and index the entire geographic area. The choice of index structures depends on the nature of the applications. The details of the index structures are given in the chapters describing the proposed models.

Definition 2 (User Activity) A user activity is made of a five tuple (u, v, lv, Wv, t) which indicates that the user u visits the POI v, located at lv and described as Wv at the time t.

Definition 3 (User Profile) for each user u, we create a user profile Du, which is a set of user activities associated with u. The dataset D used in our model consists of user profiles, that is, D = {Du: u ∈ U}. Then, given a dataset D as the union of a collection of user profiles, we aim to provide POI recommendation for users. We formulate our problem as follows.

Problem 1 (POI Recommendation) given a user activity dataset D, a target user u with his/her current location l and current time t (that is, the query is q = (u, l, t)), our goal is to recommend a list of POIs that u would be interested in. In the last decade, recommender systems have been widely studied for various applications, including music recommendation, book recommendation, paper recommendation, etc. Various techniques have been developed for effective recommendation, e.g., collaborative filtering, matrix factorization, LDA, graph-based methods, etc. However, due to the specificity of human mobility on LBSNs, it is not sufficient to apply the classic methods in recommending POIs. We will present the specific challenges of POI recommendation in LBSNs in the following section.

3. CHALLENGES

Data sparsity has been a severe challenge in many recommender systems. This is also one of the most important problems in POI recommendation. There are millions of POIs in LBSNs, but a user can only visit a limited number of them. For POI recommendation, this problem gets more severe in two important scenarios: out-of-town recommendation and sequential recommendation. Existing research on personalized POI recommendation mainly explores the geographic influence to improve the recommendation accuracy, based on the observation that the geographic proximity between spatial items affect user's check-in locations [42, 43]. Recently, there are works that further integrate social influence in LBSNs to recommend items as common interests shared by social friends [29]. In terms of the temporal effect of user check-in activities in LBSNs, most existing work only investigate the temporal cyclic patterns of check-ins [22]. To our knowledge, there are few works focusing on making recommendation in out-of-town scenario or sequential recommender systems in LBSNs.

4. LITERATURE REVIEW

With the increasing deployment and use of GPS-enabled devices, massive amounts of GPS data are becoming available. We propose a general framework for the mining of semantically meaningful, significant locations, e.g., shopping malls and restaurants, from such data. We present techniques capable of extracting semantic locations from GPS data. We capture the relationships between locations and between locations and users with a graph. Significance is then assigned to locations using random walks over the graph that propagates significance among the locations. In doing so, mutual reinforcement between location significance and user authority is exploited for determining significance, as are aspects such as the number of visits to a location, the durations of the visits, and the distances users travel to reach locations. Studies using up to 100 million GPS records from a confined spatio-temporal region demonstrate that the proposal is effective and is capable of outperforming baseline methods and an extension of an existing proposal. The problem of recommending tours to travelers is an important and broadly studied area. Suggested solutions include various approaches of points-of-interest (POI) recommendation and route planning. We consider the task of recommending a sequence of POIs that simultaneously uses information about POIs and routes. Our approach unifies the treatment of various sources of information by representing them as features in machine learning

algorithms, enabling us to learn from past behavior. Information about POIs are used to learn a POI ranking model that accounts for the start and end points of tours. Data about previous trajectories are used for learning transition patterns between POIs that enable us to recommend probable routes

5. MODULES

I.Recommender System

Each person has own preferences and personality. Recommender Systems try to use your preferences to personalize your experiences on the web and any other applications, and answer some questions like Who, Where, What etc. For example, where to eat? What to buy? Or even who you should be friends with?

- **Recommender Systems Types**

Two most well-known types of recommender systems approaches are:

- **Content Based (CB)**: The content-based approach is based on the contents, such as user pro-files, attributes of the items etc. User profile shows user preferences or as another.

Example: based on the user profile we can see which genre of movies he likes.

- **Collaborative Filtering (CF)**: The popular and most used approach can be divided into Memory-based and Model-based. The Memory-Based collaborative filtering approach can be divided into two main sections: **User-Item filtering and Item-Item filtering.**
- **Hybrid Recommender Systems**: Models that use both ratings and content features are called Hybrid Recommender Systems where both **Collaborative Filtering and Content-based Models** are combined.

I.Memory Based Collaborative Filtering

As we said before, memory-based is divided into User-Item and Item-Item. The User-Item collaborative filtering methods try to find similar users and based on the similar user's preferences (i.e., like, rating, check-in etc.) recommend new things to users. In contrast, Item-Item filtering will take an item, find users who liked that item then find other items that those users or similar users also liked. So, based on taken item it makes a recommendation. The similarity values between items in Item-Item Collaborative Filtering are measured by observing all the users who have rated both items and for User-Item Collaborative Filtering, the similarity values between users are measured by observing all the locations that are checked-in (check-in frequency) by both users. There are some distance measures, such as Cosine Similarity, Jaccard Similarity etc. The Cosine Similarity for user u and u′ is sin (u, u′) = cos (θ) =ru.ru′||ru|| ||ru′ || = ∑irui.ru′i√∑ir2ui√∑ir2u.

II. Model Based Collaborative Filtering

Model-Based CF uses machine learning and data mining approach to make a model to predict ratings. In recent years, most of the model-based approaches are based on Matrix Factorization (MF), and MF approaches received great attention because they are better than memory-based approach for scalability and sparsity. As an unsupervised learning method, the goal of the MFs is to learn the latent preferences of users and latent attributes of items based on known ratings. Actually, in this method, a model tries to learn latent or hidden features of data characteristics to predict the unknown ratings.

III. Singular Value Decomposition (SVD)

Singular Value Decomposition or in short SVD is one the well-known and most used matrix Factorization method. Collaborative Filtering or ratings can be formulated by approximating a matrix X by using singular value decomposition. The general equation can be expressed by

X = U×S×VT. Matrix X as an n× m matrix can be factorized to U, S and V: - U is an m×r orthogonal matrix (user's feature vectors in hidden feature space) - S is an r ×r diagonal matrix (singular values of X) - VT is an r × n orthogonal matrix (item's feature vectors in hidden feature space).

IV. ROOT MEAN SQUARE ERROR (RSME):

In machine learning, it is extremely helpful to have a single number to judge a model's performance, whether it is during training, cross-validation, or monitoring after deployment. Root mean square error is one of the most widely used measures for this. It is a proper scoring rule that is intuitive to understand and compatible with some of the most common statistical assumptions. By squaring errors and calculating a mean, RMSE can be heavily affected by a few predictions which are much worse than the rest. If this is undesirable, using the absolute value of residuals and/or calculating median can give a better idea of how a model performs on most predictions, without extra influence from unusually poor predictions. "Root Mean square is the standard deviation of the residuals. Now let's understand what Standard deviation and residuals are.

6. EXPERIMENTAL RESULTS

In this section, we provide an empirical evaluation of the performances of the proposed model. All the experiments were performed on a large real-world LBSN dataset collected from Foursquare, one of the largest and

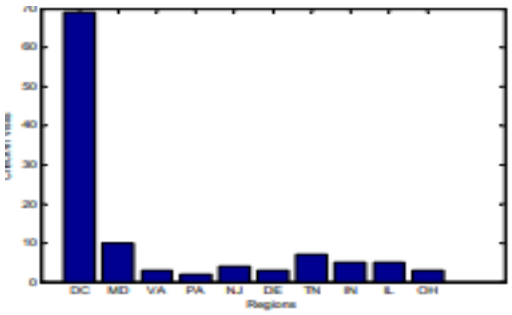


Figure 1: An example of check-in reports for a user most popular LBSN community.

1. The Experimental Data

The dataset is formulated as follows [4]: Foursquare users usually report their check-ins of POIs via Twitter. When a LNSN user posted a Tweet, which indicates a check-in of POI, we consider it as the user has checked in physically. Also, from Foursquare, we have detailed information of each POI with its location in terms of latitude and longitude, region, the associated categories, tags, the total number of people, and the total number of check-ins. With both the LBSN's tweet check-in reports, in which latitude and longitude are available, and the LBSN check-in profiles have latitude and longitude values, we can match these two sources of information to obtain LBSN users' check-in profiles with additional information for the POIs. Figure 3 shows the check-in report times for the user in different regions. A user would usually have her/his home address, which corresponds to the

highest frequent report region, and may visit POIs at different regions

Table 3: Data Description

user	POIs	rating	avg # rates	sparsity
35,025	49,779	1,080,824	30.85	99.94%

As a lot of users may have checked in or reported few check-ins, we exclude those users with less than 6 check-in records. As the number of words associated with each POI varies dramatically, we select the POIs with the minimum 10 tags. We finalized a dataset

topic	terms
1	airport terminal travel airlines delta gate tsa high mile gogo gogoinflight united southwest wifi baggage continental airplane handler airways
2	san technology apple office bar diego home iphone gym shop computer ipod store video mac coffee center ipad restaurants
3	sea seattle tac bar seatac wifi coffee beer free waterfront airport food fish limo restaurant limousine square hour colman
4	train station transit new bus subway rail metro public food nj transportation line amtrak york penn city jersey express
5	bar food beer coffee wine restaurant free bbq music burgers trivia patio pool delivery italian chicken american outdoor bon
6	theater movie movies theatre food gallery photo booth photobooth popcorn mall cinema pizza douchebag cineplex shopping imax art store
7	college university frat gas food library school pizza student coffee center state boys bar building campus store station gym
8	marketing design media social web office music corporate advertising food coffee search seo agency development restaurant internet digital toronto
9	attorney law injury accident lawyer personal lawyers attorneys city atlanta firm bar beer restaurant bankruptcy office food sports oc
10	mall store food mobile accessories shopping american wireless apple cell court phone department macy coffee photobooth body women shoes

Shown in Table 3. Here, we use implicit rating, namely the number of checks-in for POI as the rating for the POI. This is different from the rating in movie recommendation, in which the rating is usually in a range from 1 to 5. So, we need to transfer the discrete rating to a value between [0, 1] by using f(x) = (x − 1)/ (K − 1) with K is the maximum rating value [15]. We can see that the rating matrix is very spare with 99.94% missing ratings. 5.2 Evaluation Metrics we adopt the Root Mean Squared Error (RMSE) to measure the prediction error. RMSE is defined as RMSE = q 1 N P i, j (ˆrij − rij) 2 where rij denotes the rating of POI j by user i, ˆrij denotes the corresponding rating predicted by the model, and N denotes the total number of the tested rating. The smaller the value of RMSE, the more precise a recommendation is. Ranking a recommendation list is often more important than the rating prediction. First, we select the |T | highest rating set T from the test dataset. For each POI cj ∈ T for user ui, we add another |C| randomly selected POIs C, and predict the rating for {cj, C}. Then, we sort the |C| + 1 predicted rating score in a descending order. In this way, we can find the relative place of these interesting POIs in the total order of the recommendation list for a given user. We can obtain a cumulative distribution of the relative ranking based on the selected rating set T.

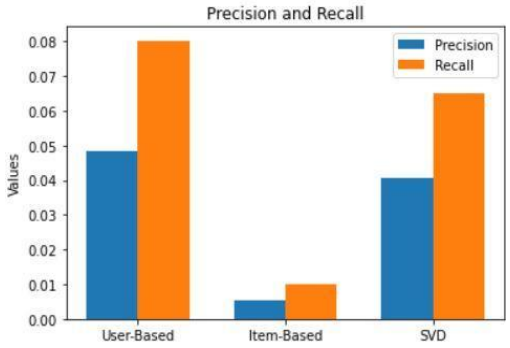


Fig: Precision and Recall

6.2 Performance Comparisons

Here, we compare the performances of different approaches in terms of RMSE and Top-N metrics. 5.4.1 Performance comparison I: RMSE with RMSE, we compare TL-PMF and PMF at different settings. We first set the number of topics K = 30 and K = 50 to learn user topic interest, and thus get the topic and location index T Lij. We do not directly use E (rij |ui, cj) = g (T Lij · U T i Cj) for prediction but pass the results through a logistic function g(x) = 1 1+exp (−x) to bound the prediction score to range [0, 1]. Then the prediction becomes: E (rij |ui, cj) = g (T Lij · U T i Cj). In each topic number case, we perform TL-PMF with different user and POI factor dimensions (D = 10 and D = 30). Also, we compare the effect of local popularity Pj in recommendation.

6.3. Summary

In summary, the proposed models can outperform the baseline method dramatically in terms of both RMSE and Top N metrics. We have observed that both personalized user interest topic as well as location dependent word-of-mouth opinion can be incorporated into the proposed flexible framework to improve recommendation performance.

6.4. Related Work

Related work can be grouped into two categories: works study place of interest recommendation from the application perspective, and works study how to exploit textual information to improve recommendation from the methodology perspective. With increasing popularity of LBSNs, applying POI recommendation to provide better location-based service has caught a lot of attentions from both academia and industry. Previous studies on POI recommendations mainly relied on user trajectory data. For example, various works [21, 2, 20, 8, 13, 7] applied collaborative filtering-based method to recommend locations and travel packages based on user trajectory data. By considering the geographical influence due to the spatial clustering phenomenon in LBSN users, Ye et al [18] explored user preference, social influence and geographical influence for recommending POIs in LBSNs. More recent work began to explore textual information to better understand patterns in LBSN and to improve LBSN services. For instance, [5] applied topic models to identify daily location-driven routines by mining text from mobile phone data. [17] Presented a work on semantic annotation for LBSNs to annotate places with category tags by exploring explicit patterns of individual places and implicit relatedness among similar places. A straightforward way is to combine collaborative filtering with topic models. By mining the textual information associated with each item, we could combine probabilistic matrix factorization (PMF) [15] and topic models [16]. The fLDA model in [1] follows this line, but they associated the rating by regularizing both user and item factors simultaneously through user features and words associated with each item. In addition to exploring topic models for item recommendation, there are also studies which use topic models to learn social-media user interests to recommend new friends with similar interests [14]. Unlike the tasks of recommending movies and scientific papers [16], the problem of POI recommendation in LBSN services is location-aware, personalized, and context depended. In addition, the textual terms associated with POIs are usually incomplete and ambiguous. This study explores both associated textual and context information to address these challenges.

7. CONCLUSION

Now, we have implemented two basic approaches for location-based recommender systems. As you saw, the Memory-based CF algorithms are easy to implement and the result has appropriate prediction quality. The main challenges in Memory-based approaches are scalability and cold-start problems. They are not suitable for a real-world scenario, and they have problems when a new user or item is added to the system (i.e., cold-start). In contrast, Model-Based CF approaches are scalable and can handle when data is sparse but also suffer from the cold-start problem. Also, more recent work in the Model-based CF minimizes the squared error by applying alternating least square or stochastic gradient descent and uses regularization terms to

prevent over fitting. In various recommendation systems, location information about items and users helps to generate the recommendation results. Location data helps to link the physical and digital domain and allows collecting detailed knowledge about particular user preferences and activities. Location is a vital concept in recommendation systems which specify the relationship between users, between locations and between users and location. Various recommendation systems use location factor as a major component to find out the correct and accurate results. Location information is considered as a context in various content based and contextual recommendation techniques. Location information is useful to calculate the accurate and effective recommendations in different recommendation systems.

REFERENCES:

[1] Jun Zeng, Feng Li, Haiyang Liu, Junhao Wen, Sachio Hirokava, "A Restaurant Recommender System Based on User Preference and Location in Mobile Environment," in 5th IIAI International Congress on Advanced Applied Informatics, 2016.

[2] Kasra Madadipouya, "A Location based Movie Recommender System using Collaborative Filtering," in International Journal in Foundations of Computer Science & Technology (IJFCST), Vol.5, No.4, July 2015.

[3] Yuichiro Takeuchi, Masanori Sugimoto, "An Outdoor Recommendation System based on User Location History," in ubiPCMM, 2005.

[4] Wan-Shiou Yang, Hung-Chi Cheng, Jia-Ben Dia, "A location-aware recommender system for mobile shopping environments," in Expert Systems with Applications, 437–445, Elsevier, 2008.

[5] Zhiwen Yu, Miao Tian, Zhu Wang, And Bin Guo, "ShopType Recommendation Leveraging the Data from Social Media and Location-Based Services," in ACM Transactions on Knowledge Discovery from Data, Vol. 11, No. 1, Article 1, July 2016.

[6] Hongzhi Yin, Bin Cui, Yizhou Sun, Zhiting Hu, Ling Chen, "LCARS: A Spatial Item Recommender System," in ACM Trans. Inf. Syst. V, N, Article A, 35 pages, 2014.

[8] Tomas Horvath, "A Model of User Preference Learning for Content-based Recommender Systems," in Computing and Informatics, Vol. 29, 1001–1029, 2008.

[9] Jie Bao, Yu Zheng, David Wilkie, and Mohamed Mokbel "Recommendations in location-based social networks: A survey," in GeoInformatica, 525565, 2015.

[10] Jun Zeng, Feng Li, Haiyang Liu, Junhao Wen, Sachio Hirokava, "A Restaurant Recommender System Based on User Preference and Location in Mobile Environment," in 5th IIAI International Congress on Advanced Applied Informatics, 2016.

[11] Kasra Madadipouya, "A Location based Movie Recommender System using Collaborative Filtering," in International Journal in Foundations of Computer Science & Technology (IJFCST), Vol.5, No.4, July 2015.

[12] Yuichiro Takeuchi, Masanori Sugimoto, "An Outdoor Recommendation System based on User Location History," in ubiPCMM, 2005.

[13] Wan-Shiou Yang, Hung-Chi Cheng, Jia-Ben Dia, "A location-aware recommender system for mobile shopping environments," in Expert Systems with Applications, 437–445, Elsevier, 2008.

[14] Zhiwen Yu, Miao Tian, Zhu Wang, And Bin Guo, "ShopType Recommendation Leveraging the Data from Social Media and Location-Based Services," in ACM Transactions on Knowledge Discovery from Data, Vol. 11, No. 1, Article 1, July 2016.