ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN FACULTY OF ELECRICAL ENGINEERING
ELECTRONIC AND COMMUNICATION SYSTEMS INSTITUTE
INSTRUMENTATION AND AUTOMATION DEPARTMENT

FINAL ESSAY

OE-KVK **Aibek Abdikazakh**

2023. **T001234/FI12204/K**

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Óbuda University
Kandó Kálmán Faculty of Electrical Engineering

Institute of Instrumentation and Automation

THESIS ASSIGNMENT

Student's surname, forename (s): Aibek Abdikazakh
Thesis number: SZD2309040923466635[redacted]
Registration number: T009653/FI12904/K **Neptun code:** [redacted]

Branch of study:
Specialization: Instrumentation and automation specialisation

Thesis title: Recommendation systems: overview, technologies and a practical application project
Task description: Exploration of the recommendation systems operation: basic concepts and most common approaches for recommendation system design. Usage overview of different technologies effective in the context of recommendation system development and improvement such as reinforcement learning and deep learning.
Recommendation system project development and its optimization.

Institutional consultant's name: Döníz Borsos

The limitation period of the theme issued: 31.12.2025.
Deadline for submission of Thesis: 15.12.2023.

The thesis is: Not secreted.

Issued: Budapest, 24.10.2023. P.H .....
Director of Institute

The Thesis is suitable for submission:

.....
Institutional consultant

.....
External consultant

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

KANDÓ KÁLMÁN FACULTY OF ELECTRICAL ENGINEERING
ELECTRONIC AND COMMUNICATION SYSTEMS INSTITUTE
INSTRUMENTATION AND AUTOMATION DEPARTMENT

STUDENT STATEMENT

I hereby declare that the present final thesis paper is solely my own work and that the references (written and electronic literature and tools used during the process) have been acknowledged and fully cited. I hereby grant the university and the task assigning institution permission to use free of charge the final results of this paper for their own purposes, abiding by the pertaining rules of encryption.

Budapest, 2023.12.14

.....
Signature of student

Table of Contents

Abstract

In the era of information overload, online platforms accumulate vast amounts of user-generated content, including reviews of various places and establishments. Leveraging this valuable data, content-based recommendation systems have emerged as pivotal tools for personalized user experiences. This paper investigates the recommendation algorithms, with a specific focus on utilizing textual content from reviews to offer personalized suggestions to users.

The research begins by exploring the challenges posed by traditional recommendation systems and the advantages offered by content-based approaches. It delves into natural language processing techniques, sentiment analysis, and machine learning algorithms to extract meaningful insights from textual reviews. The study further investigates the integration of user preferences, contextual information, and location-based data to enhance the accuracy and relevance of recommendations.

The results demonstrate the system's ability to provide personalized and contextually relevant recommendations, thereby enhancing user satisfaction and engagement.

This research contributes to the evolving landscape of recommendation systems, offering valuable insights into the fusion of textual content analysis and machine learning techniques. The findings have significant implications for online platforms, businesses, and consumers, paving the way for more effective and user-centric content-based recommendation systems in the digital realm.

1. INTRODUCTION

In the digital age, the overwhelming amount of information has compelled online platforms to employ sophisticated recommendation systems, ensuring users are presented with content tailored to their preferences. Social media platforms curate posts and ads based on user's interests and location, while e-commerce websites utilize search history, demographics, and buying patterns to showcase products tailored to individual preferences. This approach resolves numerous challenges for both businesses and users.

By eliminating the need to sift through an extensive catalog of options, these systems save users valuable time and enhance their engagement with relevant products, fostering a more efficient decision-making process [1]. Moreover, in the realm of e-commerce, recommender systems not only streamline the user experience but also significantly boost revenues by increasing sales [2]. Beyond commerce, these systems extend their benefits to users by helping them discover new options in the form of books, movies, and songs they might have otherwise missed.

This thesis aims to unravel the intricacies of recommender systems, particularly those fueled by textual data, with a specific emphasis on their application in recommending establishments, such as restaurants and cafes. While numerical ratings and categorical preferences form the foundation for recommendations, the examination of textual reviews in the context of establishment recommendations presents a unique challenge due to the limited availability of standardized data. In this realm, platforms like Google Maps offer basic information such as location and opening hours; however, the richness of valuable insights resides within the detailed textual descriptions and reviews provided by users.

Understanding the practical challenges faced by recommender systems in recommending food establishments, this research seeks to explore both the theoretical and practical aspects of their application. By delving into the complexities of processing textual data and extracting meaningful information, the aim is to refine and enhance the recommendation process, providing users with more tailored and relevant suggestions. The focus on establishments acknowledges the distinctive challenges posed by the subjective nature of user opinions and the need for nuanced contextual understanding in this domain.

Through a comprehensive exploration of recommender systems, this thesis endeavors to bridge the gap between the theory and practical implementations. By addressing the inherent challenges arising from varied data sources and limited standardization in the establishment recommendation context, the research aims to propose innovative solutions. The overarching objective is to contribute to the evolution of recommendation technology, making it more adaptable, nuanced, and attuned to the dynamic needs of users in both theoretical and practical applications, particularly in the context of suggesting places of interest such as restaurants and cafes.

2. Recommender systems types

This chapter delves into recommender systems types, facilitating an understanding of the different types and their operations. It is essential to grasp the basic knowledge of recommender systems, in order to explore the practical applications of these systems in real-life scenarios. By providing a comprehensive understanding of these systems, this chapter lays a foundation for the subsequent chapters, which will examine practical examples and applications.

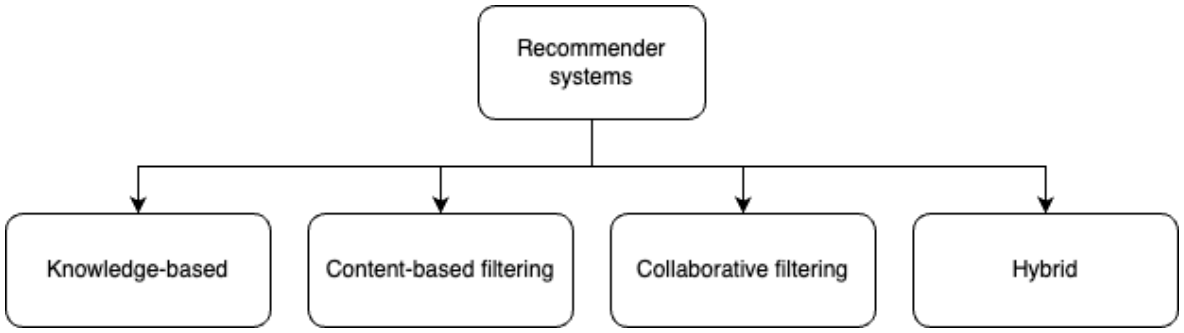


Figure 1. Recommender systems types

2.1 Knowledge-based recommender systems

Knowledge-based recommender systems operate on explicit knowledge about users, items, and recommendation criteria [3]. They prove invaluable in situations where other recommendation systems falter, especially when items are infrequently purchased, leading to insufficient data for feature learning. In these instances, Knowledge-based recommenders take a proactive approach by directly querying users about their preferences, utilizing this firsthand information to initiate tailored recommendations.

These systems excel in overcoming the daunting "cold start" problem, a scenario where there's insufficient gathered data about a user to draw meaningful inferences. By relying on user-provided preferences, knowledge-based recommenders navigate this challenge, ensuring personalized recommendations even in the absence of extensive user history. Their ability to seamlessly adapt to such circumstances highlights the effectiveness of knowledge-based approaches in enhancing user experience and satisfaction.

2.2 Content-based filtering recommender systems

Content-based filtering is a method that focuses on analyzing item attributes to predict user preferences. By examining a user's past evaluations, the system suggests items similar to those the user has liked before [4][5].

It uses different probabilistic models such as Naïve Bayes Classifier, Decision Trees, and Neural Networks to find similarities between items and offer relevant suggestions.

Content-based filtering does not rely on other users' profiles, which could be beneficial if the user database is limited. It quickly adjusts to changes in a user's preferences, making it responsive.
However, it requires a deep understanding and description of the items' features in the user's profile. [5]

2.2.1 Content-based systems advantages and disadvantages

Content-based (CB) filtering techniques address the limitations of Collaborative Filtering (CF) by offering solutions to several challenges. Unlike CF, CB filtering can recommend new items even if users haven't rated them. This means that even if there's no information about user preferences in the database, the accuracy of recommendations isn't affected. Additionally, CB filtering can quickly adapt its suggestions if a user's preferences change. It can handle situations where users don't share the same items but have similar intrinsic features. One notable advantage is that users can receive recommendations without sharing their personal profiles, ensuring privacy [6]. CB filtering techniques can also provide explanations to users about how recommendations are generated.

However, CB filtering techniques come with their own set of issues. They rely heavily on detailed metadata of items and well-organized user profiles before making recommendations. This requirement is known as limited content analysis, and the effectiveness of CB filtering depends on the availability of descriptive data. Another

problem is content overspecialization, where users are limited to receiving recommendations similar to items already defined in their profiles.

2.3 Collaborative filtering recommender systems

Collaborative filtering is a method used to predict and recommend items like movies or music to users, even when these items do not have detailed descriptions or metadata. It works by creating a database that shows how users rate different items. Based on this data, the technique finds users with similar tastes and preferences called a "neighborhood" and suggests items to a user that they had not rated before but were liked by others in their neighborhood. These suggestions can be numerical predictions, expressing the expected score for an item, or a list of top recommended items. [5]

2.3.1 Collaborative Filtering Systems Advantages and Disadvantages

Collaborative Filtering (CF) offers several advantages over Content-Based Filtering (CBF). One major strength of CF is its ability to function in domains where items lack substantial associated content or where the content is challenging for a computer system to analyze, such as opinions or ideals. Additionally, CF can provide serendipitous recommendations, suggesting items that are relevant to the user even if those items are not present in the user's profile. CF techniques determine recommendations by considering similarities between users and items simultaneously, and they automatically learn feature representations. This means there is no requirement to create specific features manually or possess extensive domain knowledge about those features.

Cold-start problem

This problem arises when a recommender system lacks sufficient information about a new user or item to make accurate predictions. New users or items have empty profiles because they haven't rated any items yet, making it difficult for the system to understand their preferences.

Data sparsity problem

This issue occurs when only a small fraction of items in a database are rated by users, leading to a sparse user-item matrix [4]. Sparse data makes it challenging to identify suitable neighbors for recommendations, resulting in weak and limited suggestions. Data sparsity also affects the coverage of recommendations, indicating the percentage of items the system can provide recommendations for.

Scalability

Scalability problems emerge as recommendation algorithms struggle with increased computational demands proportional to the growing number of users and items. Techniques efficient for smaller datasets may fail to generate satisfactory recommendations when dealing with larger volumes of data. It is essential to employ recommendation methods that can scale effectively as the dataset size expands. Solutions involve using Dimensionality reduction techniques like Singular Value Decomposition (SVD) to produce reliable and efficient recommendations.

Synonymy

Synonymy refers to similar items having different names or entries, posing a challenge for recommender systems. Distinguishing between closely related items, such as "baby wear" and "baby cloth," can be difficult. Collaborative Filtering systems often struggle to compute their similarity. Various methods, including automatic term expansion, the creation of a thesaurus, and techniques like Latent Semantic Indexing, can address synonymy issues. However, these methods may introduce terms with unintended meanings, leading to a decline in recommendation accuracy [5].

2.4 Hybrid recommender systems

Hybrid recommender systems utilize a blend of two or more recommendation techniques to enhance performance while minimizing the drawbacks associated with individual methods [4]. The fundamental concept behind hybrid systems is that a combination of algorithms can yield more precise and efficient recommendations compared to a single algorithm. This approach leverages the strengths of one algorithm to compensate for the weaknesses of another, resulting in improved overall accuracy and effectiveness.

2.4.1 Deep Reinforcement Learning in recommender systems

In contemporary recommendation systems, the integration of deep reinforcement learning (DRL) stands out as a frequent component within a hybrid recommender system.
DRL excels in optimizing recommendations through the lens of user interaction metrics, by using the interaction metrics as a reward signal to optimize a personalized policy that adapts to individual preferences. This hybrid system, coupling DRL with other recommendation techniques like collaborative filtering and content-based filtering, overcomes the limitations of individual methods, delivering recommendations that are more accurate, diverse and more specific to the individual user. DRL's inherent ability to address the exploration-exploitation tradeoff ensures a delicate balance in recommending familiar items while exploring new suggestions to accommodate shifting user preferences. Furthermore, its application mitigates cold start issues, particularly relevant for new users or items, by drawing insights from historical data and continually adapting recommendations based on real-time user feedback. In essence, the amalgamation of DRL in hybrid recommendation systems not only offers dynamic and adaptive recommendations but also addresses critical challenges such as the exploration-exploitation tradeoff and cold start issues, presenting a comprehensive solution to enhance the efficacy of recommendation systems [7].

3. Recommendation Systems Practical Application - Restaurants and Establishments Recommender

This section of the thesis demonstrates the practical application of recommender systems for restaurant and establishment recommendations. Google Maps data and user preferences for restaurants will be used to create a recommender system that delivers the most relevant restaurant suggestions to the user.

3.1 Dataset

The dataset used to build the content-based recommendation system contains comprehensive information about the places, including details like opening hours, addresses, coordinates, and numerous other parameters. These additional data points can be utilized to create context-aware recommendations tailored to specific user preferences and requirements.

Place ID	Name	Country	City	Postal Cc	Sublocality	Route	Stree	Latitude	Longitude	Editorial S	Editi	Rating	Reviews Count	Pric	Dine-in	Monday_(
ChIJew3oD1vcQUcR88sk5uM2	All About Street Food	Hungary	Budapest	1088	VIII. kerület	Múzeum utca	7	47.490103	19.0633697			3.9	144	2	TRUE	1100
ChIJJzNmdZLdQUcROh6kUWa	Street food	Hungary	Budapest	1042	V. kerület	József Attila utca	31	47.499183	19.052779			4.7	22		TRUE	
ChIJew3oD1vcQUcR88sk5uM2	All About Street Food	Hungary	Budapest	1088	VIII. kerület	Múzeum utca	7	47.490103	19.0633697			3.9	144	2	TRUE	1100
ChIJkd1d4mrcQUcR53ZWIBrMI	Meatology Budapest	Hungary	Budapest	1065	VI. kerület	Bajcsy-Zsilinszky (	15/b	47.500833	19.0552457			4.6	3252	2	TRUE	1130
ChIJJzNmdZLdQUcROh6kUWa	Street food	Hungary	Budapest	1042	V. kerület	József Attila utca	31	47.499183	19.052779			4.7	22		TRUE	
ChIJw0l9AfXdQUcR6vnsbvmUc	Fit Fast Food	Hungary	Budapest	1074	VII. kerület	Dohány utca	48	47.497372	19.0679365			4.5	2		TRUE	
ChIJ8z64GGjcQUcRDtjy8jgvJIo	Street Food Karavan Budapest	Hungary	Budapest	1075	VII. kerület	Kazinczy utca	18	47.497242	19.0632554	Burgers, c	en	4.3	8729		TRUE	1130
ChIJ5959IZDdQUcRIkz4FBUgH	Papitos Mexican Street Food - Mad	Hungary	Budapest	1075	VII. kerület	Madách Imre út	5	47.497786	19.0578966			4.7	584		TRUE	1100
ChIJQZ_OxNfdQUcRg0h1M2rI9	Papitos Mexican Street Food - Dohá	Hungary	Budapest	1074	VII. kerület	Dohány utca	68	47.498386	19.0694911			4.6	561		TRUE	1100
ChIJ2eWPmRXdQUcRRFGehEE	MINO'S	Hungary	Budapest	1066	VI. kerület	Teréz körút	10	47.504480	19.0639552			4.8	57		TRUE	1000
ChIJW0ddTg3cQUcRAJq58R6H	CheChe Bistrocafe	Hungary	Budapest	1066	VI. kerület	Teréz körút	56	47.509002	19.0576431			4.4	903		TRUE	1100
ChIJ40vavMjdQUcRvvQ3b35bR	Street food 1:	Hungary	Budapest	1075	VII. kerület	Dob utca	2	47.496273	19.0589894						TRUE	
ChIJW_xUr0LcQUcR-feyVQrPtC	Bors Gastro Bar	Hungary	Budapest	1075	VII. kerület	Kazinczy utca	10	47.496726	19.0635487	Trendy bis	en	4.8	5586		TRUE	1130
ChIJ1ek1d2jcQUcR1wtOR6mhe	Falafel Bar	Hungary	Budapest	1077	VII. kerület	Wesselényi utca	30	47.498533	19.0653285			4.4	1233	1	TRUE	1100
ChIJy9fmK3ndQUcRlpns5j7feT(	Tarján Food Kft.	Hungary	Budapest	1077	VII. kerület	Bethlen Gábor utc	1	47.501033	19.0819041						TRUE	600
ChIJc2-xGh_dQUcRDLQRJhxq7	Syran Fast Food	Hungary	Budapest	1074	VII. kerület	Dohány utca	39	47.497439	19.0684584			4.6	36		TRUE	1200
ChIJG6PrTgrcQUcRX7DxZzzBC	Street Food 13 ker	Hungary	Budapest	1132	XIII. kerület	Csanády utca	4b	47.516054	19.058304			4.5	527		TRUE	1000
ChIJedpUapndQUcRzD1I8lRhc	Piac New Hungarian Food Hall	Hungary	Budapest	1056	V. kerület	Váci utca	36	47.492229	19.0538116						TRUE	
ChIJA2yh7xPcQUcREFpkRMnd	Budapest Bistro	Hungary	Budapest	1054	V. kerület	Vécsey utca	3	47.505334	19.0492944			4.4	2052	3	TRUE	1130
ChIJEx2KqNDdQUcRnp_zqf-4h	Ke La La Wok	Hungary	Budapest	1091	IX. kerület	Üllői út	113	47.480311	19.0852348			4.5	87		TRUE	1000

Figure 2. An example of some of the establishment details dataset entries.

Pandas, a Python library for data manipulation and analysis, is used to transform and analyze the dataset.

Figure 3. Pandas command that shows the dataset shape.

The dataset contains 6676 rows and 40 columns, or 6676 samples and 40 features.

Figure 4. The full list of the datasets' features.

When recommending places to users, two primary approaches, content-based filtering and collaborative filtering, can be employed. In this case, the preference is for content-based

filtering because the dataset includes detailed descriptions of items (places) but lacks sufficient user information. Content-based recommenders treat the recommendation task as a user-specific classification problem. They develop a classifier to discern a user's likes and dislikes based on the features of items. Consequently, users in our recommender system can specify a list of preferred and disliked places, receiving pertinent recommendations tailored to their preferences.

The system generates a content-based user profile by forming a weighted vector of item features. These weights indicate the significance of each feature to the user and can be derived through various methods. Basic methods involve calculating the average values of the rated item vector, while more advanced techniques leverage machine learning methods like Bayesian Classifiers, cluster analysis, decision trees, and artificial neural networks. These approaches aim to estimate the likelihood that the user will have a favorable opinion of the item.

3.2 Data Preprocessing

Data preprocessing is a crucial initial step in machine learning projects, addressing the challenges posed by raw, often imperfect data. It involves handling noises, addressing missing values, and formatting data to align with machine learning model requirements. Data is seldom pristine in real-world scenarios, and preprocessing ensures the data is clean and structured appropriately. This removes impediments and significantly enhances the accuracy and efficiency of subsequent machine learning models. The process is indispensable for transforming raw data into a format that optimally supports the success of machine learning algorithms.

3.2.1 Handling Missing Values

In real-world datasets, encountering missing values is a common occurrence rather than an exception. This prevalence of missing data underscores the importance of addressing this issue in the data preprocessing step of machine learning projects. Handling missing values

becomes crucial as it can have a significant impact on the quality and reliability of the subsequent machine learning models. Failing to appropriately manage missing data may lead to skewed insights, biased predictions, or even compromised model performance. Thus, an effective strategy for handling missing values is essential to ensure the robustness and accuracy of machine learning analyses.

Visualizing Missing Data

Visualizing missing data is crucial for handling it because it provides a clear picture of where the gaps exist in the dataset. Using graphs or charts to represent missing values, one can easily identify patterns, clusters, or trends in the missing data. This visualization helps make informed decisions about handling missing values effectively. It allows for seeing if certain features have a higher percentage of missing data or if there's a relationship between missing values in different columns. Understanding the distribution of missing data aids in choosing appropriate strategies for imputation, deletion, or other methods, ensuring that the dataset is correctly prepared for analysis or machine learning tasks.

Numerical Summaries

Numerical summaries in missing data visualization serve as a straightforward and effective way of presenting key information about the extent and distribution of missing values in a dataset. These summaries offer simplicity in providing quantitative metrics, such as the percentage of missing data for each feature and total counts of missing values.

```
df.isnull().sum()
```

```
Place ID                          1
Name                              1
Country                           1
City                             39
Postal Code                       2
Sublocality                    1208
Route                           317
Street Number                   634
Latitude                          1
Longitude                         1
Editorial Summary              6164
Editorial Summary Language     6164
Rating                          522
Reviews Count                   522
Price Level                    4175
Dine-in                        2154
Monday_Open                    1950
Monday_Close                   1950
Tuesday_Open                   1597
Tuesday_Close                  1597
Wednesday_Open                 1462
Wednesday_Close                1462
Thursday_Open                  1412
Thursday_Close                 1412
Friday_Open                    1374
Friday_Close                   1374
Saturday_Open                  1848
Saturday_Close                 1848
Sunday_Open                    2810
Sunday_Close                   2810
Serves Beer                    3561
Serves Breakfast               4879
Serves Brunch                  4960
Serves Dinner                  3975
Serves Lunch                   3600
Serves Wine                    3922
Takeout                        2497
Types                             1
User Ratings Total              522
Wheelchair Accessible Entrance 3599
dtype: int64
```

Figure 5. Number of missing values per feature column.

It is possible to employ a custom-written function for outputting the numerical summary in a more suitable for a specific use-case format.

```
def get_numerical_summary(df):
    total = df.shape[0]
    missing_columns = [col for col in df.columns if df[col].isnull().sum() > 0]
    missing_percent = {}
    for col in missing_columns:
        null_count = df[col].isnull().sum()
        per = (null_count/total) * 100
        missing_percent[col] = per
        print("{} : {} ({}%)".format(col, null_count, round(per, 3)))
    return missing_percent
```

Figure 6. A function for calculating the number and percentage of missing values per feature column.

The number and percentage of missing values for each column can be examined using the custom-written numerical summary function.

```
missing_percent = get_numerical_summary(df)
```

```
Place ID : 1 (0.015%)
Name : 1 (0.015%)
Country : 1 (0.015%)
City : 39 (0.584%)
Postal Code : 2 (0.03%)
Sublocality : 1208 (18.095%)
Route : 317 (4.748%)
Street Number : 634 (9.497%)
Latitude : 1 (0.015%)
Longitude : 1 (0.015%)
Editorial Summary : 6164 (92.331%)
Editorial Summary Language : 6164 (92.331%)
Rating : 522 (7.819%)
Reviews Count : 522 (7.819%)
Price Level : 4175 (62.537%)
Dine-in : 2154 (32.265%)
Monday_Open : 1950 (29.209%)
Monday_Close : 1950 (29.209%)
Tuesday_Open : 1597 (23.922%)
Tuesday_Close : 1597 (23.922%)
Wednesday_Open : 1462 (21.899%)
Wednesday_Close : 1462 (21.899%)
Thursday_Open : 1412 (21.15%)
Thursday_Close : 1412 (21.15%)
Friday_Open : 1374 (20.581%)
Friday_Close : 1374 (20.581%)
Saturday_Open : 1848 (27.681%)
Saturday_Close : 1848 (27.681%)
Sunday_Open : 2810 (42.091%)
Sunday_Close : 2810 (42.091%)
Serves Beer : 3561 (53.34%)
Serves Breakfast : 4879 (73.083%)
Serves Brunch : 4960 (74.296%)
Serves Dinner : 3975 (59.542%)
Serves Lunch : 3600 (53.925%)
Serves Wine : 3922 (58.748%)
Takeout : 2497 (37.403%)
Types : 1 (0.015%)
User Ratings Total : 522 (7.819%)
Wheelchair Accessible Entrance : 3599 (53.91%)
```

Figure 7. Number and percentage of missing values per feature column.

Matrix Visualization

Matrix visualization of missing values, often represented as a nullity matrix, is a potent tool for comprehending the distribution of missing data throughout a dataset. The nullity matrix provides a comprehensive view where rows symbolize data points, columns symbolize features, and each cell indicates the presence or absence of missing values. Distinct colors or symbols are employed to highlight missing values, facilitating the identification of their prevalence and patterns across the dataset. This visualization in the

form of a sparkline or a striped line, offers a graphical emphasis on rows with the highest and lowest nullity, aiding in the identification of critical patterns and trends.

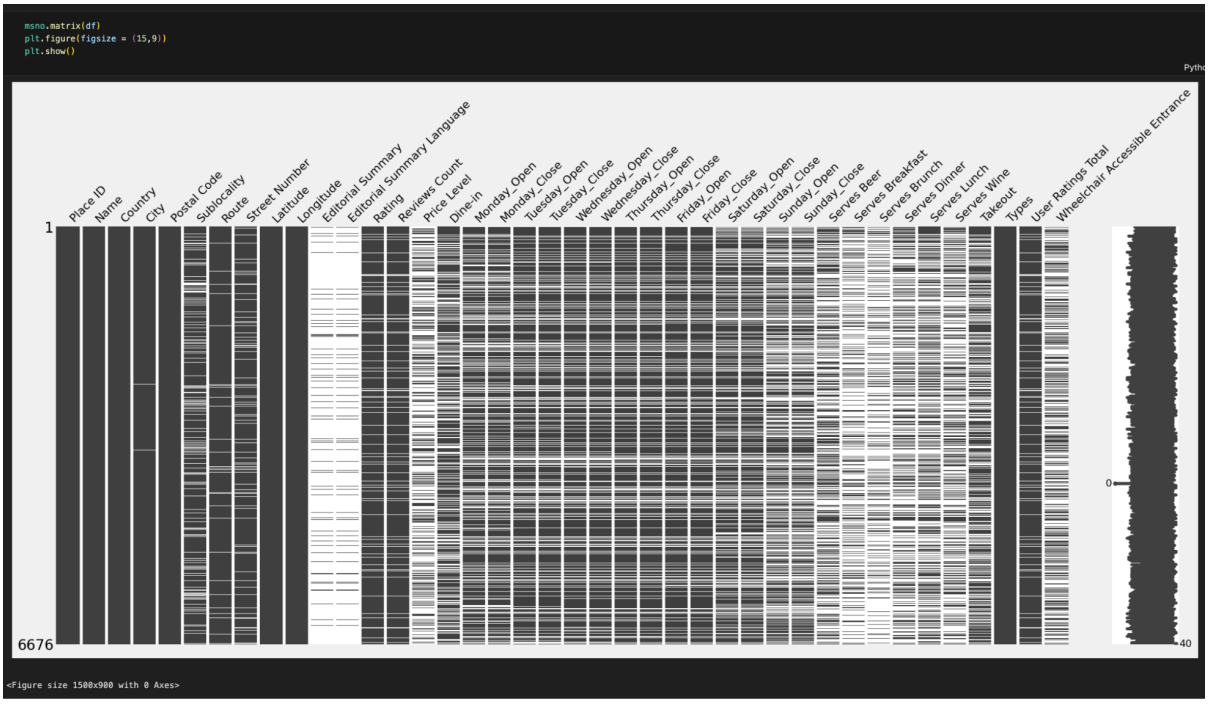


Figure 8. Matrix visualization of the missing dataset values.

Correlation Heatmap

The correlation heatmap assesses nullity correlation, indicating how the presence or absence of one feature in the dataset influences another. The nullity correlation scale spans from -1 to 1: -1 signifies that the presence of one column ensures the almost certain absence of the other, 0 implies no dependence, and one indicates that the presence of one column ensures the presence of the other. Notably, the heatmap reveals correlations in data completeness between attribute pairs. However, columns that are consistently full or empty lack meaningful correlation and are excluded from the visualization. While effective for attribute pair correlations, the heatmap has limitations in capturing broader relationships and lacks specialized support for extensive datasets.

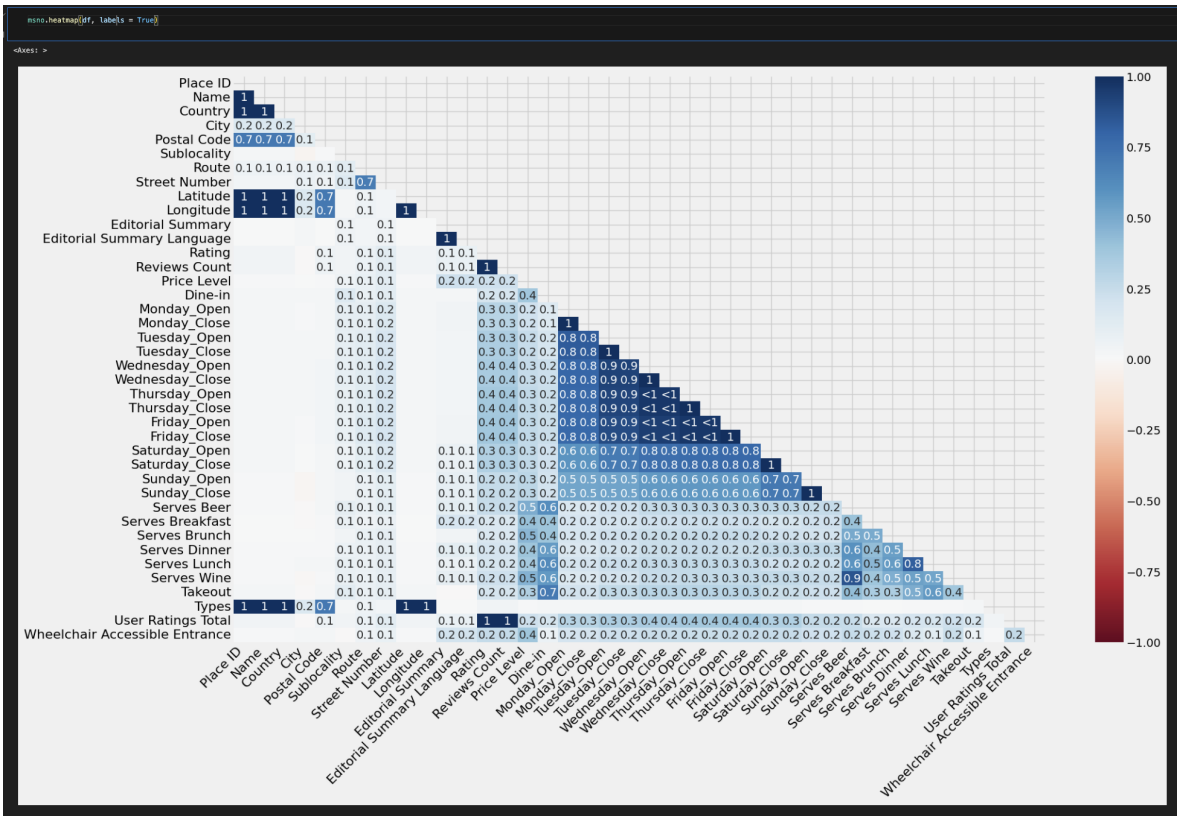


Figure 9. Correlation heatmap of the missing dataset values.

Dendogram

A dendrogram is a tree-like diagram used to display hierarchical relationships between sets of data. In the context of visualizing missing values, a dendrogram is often employed to show the hierarchical clustering of variables based on their patterns of missing data. Each variable is represented as a leaf node in the tree, and the branches of the tree depict the similarity in missing value patterns between variables. Dendrograms can provide insights into the structure of missing data, highlighting groups of variables that exhibit similar patterns of missingness.

In interpreting the dendrogram, a top-down approach is employed, where attention is given to the height at which two columns are joined, signifying nullity correlation. A lower height indicates a stronger correlation, while a greater height suggests a weaker relationship. Regarding the example provided, if a pair of attributes such as "Reviews Count" and "Rating" are joined at height 0, it indicates a high correlation in terms of

nullity. On the other hand, when attributes such as "Rating" and "Serves Brunch" have a maximum height, it signifies a lower correlation between them.

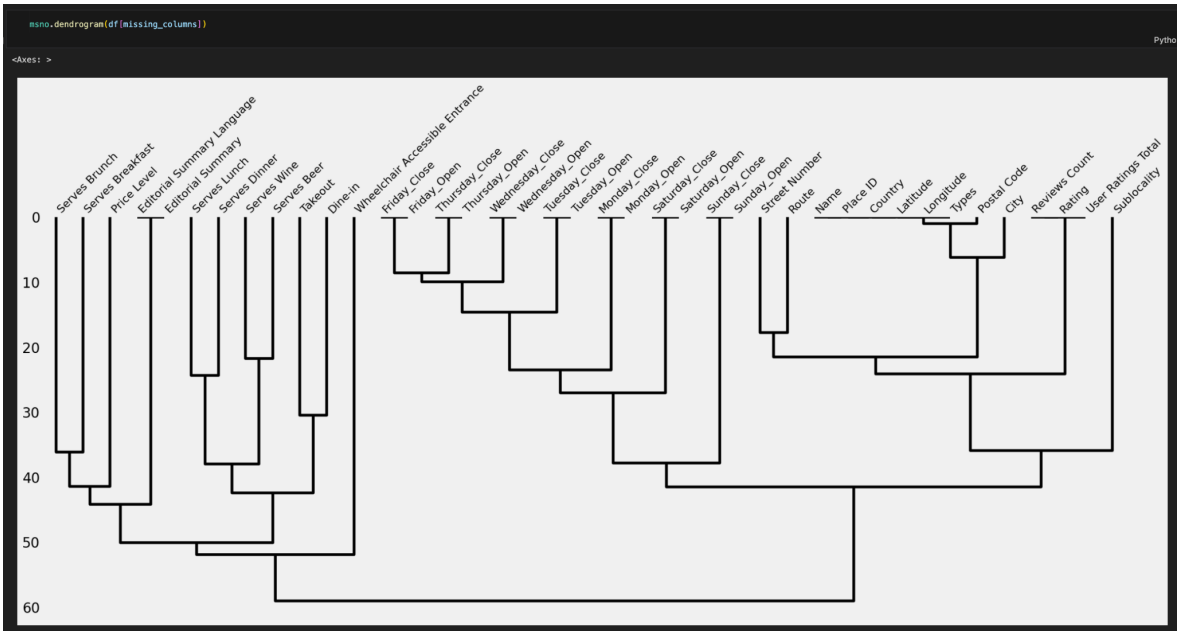


Figure 10. Dendogram representation of the missing values in the dataset.

Deletion of Data

One straightforward approach for handling missing values is to eliminate entire attributes or samples containing missing values. However, this decision should be made cautiously, taking various aspects of the data into account. For instance, one can remove predictors with any missing values or remove samples that have any missing values.

Deletion of Attributes

If an attribute has a high percentage of missing values (around 80-90%), it could be beneficial to delete that attribute instead of predicting values based on the remaining 10-20% of the data.

```
# Threshold to remove attribute having missing values greater than it
MISSING_VALUES_THRESHOLD = 90

for col, missing_percentage in missing_percent.items():
    if missing_percentage > MISSING_VALUES_THRESHOLD:
        df.drop(col, axis = 1, inplace = True)
```

✓ 0.0s

Figure 11. A script for deleting attributes with a percentage of missing values greater than a specified threshold.

Executing the script to remove attributes with a missing value percentage exceeding a set threshold of 90% resulted in the elimination of two columns from the dataset: Editorial Summary and Editorial Summary Language. Both columns had 92.331% of their values missing.

Given the paper's focus on constructing a recommender system for restaurants within a specific country, the inclusion of the "Country" column doesn't contribute significantly to machine learning algorithms. Typically, this column is expected to contain a single value and can be safely excluded from the dataset. In this specific case, examining unique values in the "Country" column revealed an additional entry, "Slovakia." This unexpected result might be attributed to Google Maps fetching restaurants based not on country but on proximity to a specified coordinate point. Given that Slovakia is a neighboring country to Hungary, it's plausible that some restaurants located in close proximity to a coordinate set in downtown Budapest were inadvertently included. After filtering out restaurants not situated in Hungary, the "Country" column can be confidently removed, as it adds no further meaningful insights to the restaurant data. Thus, additional 56 samples and 1 column were removed from the dataset.

Figure 12. Filtering restaurants that are not located in Hungary, dropping “Country” column.

Deletion of Samples

If the number of samples (M) is large for your task, deleting samples might be a viable option. However, if a sample has only a few missing values concerning attributes, other methods to fill those missing values should be considered.

Regarding the dataset, rows with less than 8 out of 37 columns were eliminated, resulting in the removal of 14 rows. This addressed incorrect samples where all values were missing. This action addressed incorrect samples where all values were missing.

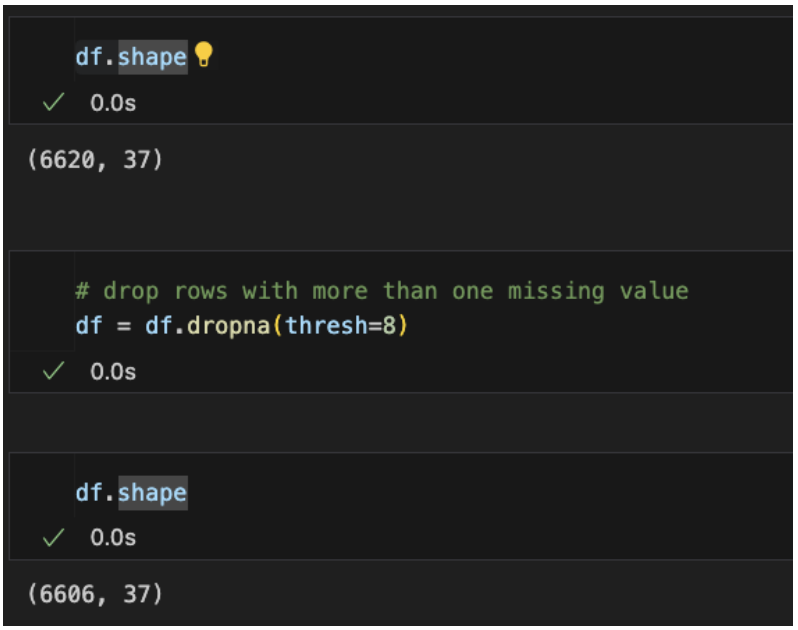


Figure 13. Dropping samples with with less than 8 present columns.

Encoding Missing Values

Encoding missing values involves representing them in a structured form that can be handled by machine learning algorithms. One common approach is to assign a specific placeholder or label to denote missing values in the dataset. This ensures that the missing values are recognized by the algorithm during data processing.

For categorical features, missing values can be encoded as a separate category or labeled with a unique identifier, allowing the algorithm to distinguish them from valid categories. This prevents the loss of information related to the absence of certain values.

In numerical features, missing values can be encoded as a specific number (e.g., -1) or using statistical measures like the mean or median of the existing values. This maintains the numerical structure of the data while providing a clear indication of missing information.

By using Pandas, it can be determined which columns of the dataset have missing values to perform operations exclusively on them.

Figure 14. A list of columns where at least 1 value is missing.

Additionally, the type of the columns can be determined as well, in case separate operations need to be performed on columns with different value types.

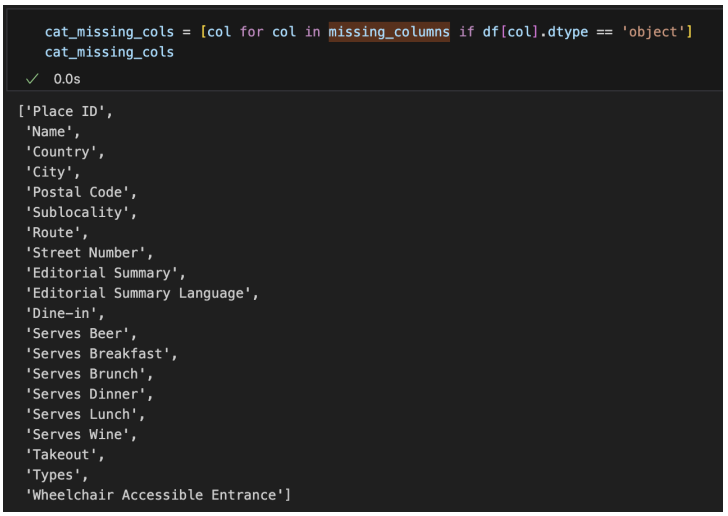


Figure 15. A list of categorical columns with missing values.

One possible way to encode missing categorical values is to fill all the missing values with a new category, e.g. “Missing”.

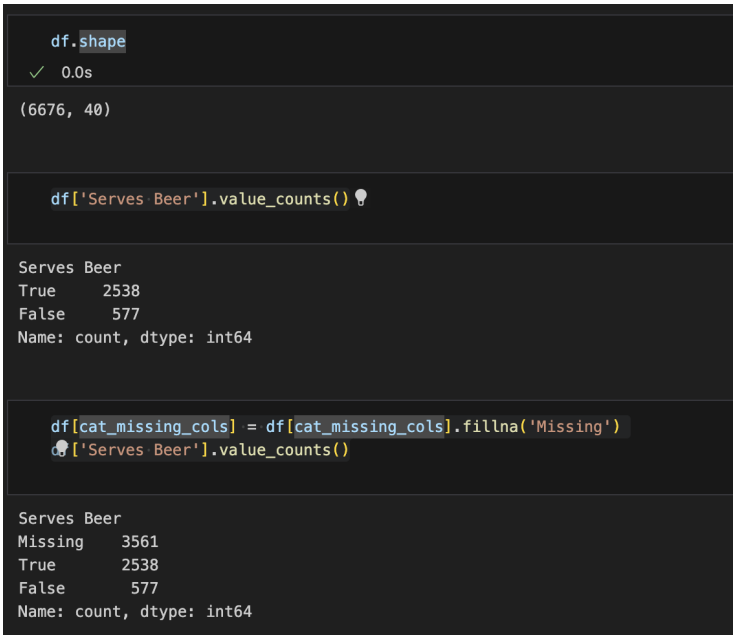


Figure 16. An example of Missing Values Encoding.

One can obtain information about missing values in columns before and after encoding by utilising Pandas' info method. Upon comparing the two sets of information, it becomes evident that all the missing values have been successfully filled in.

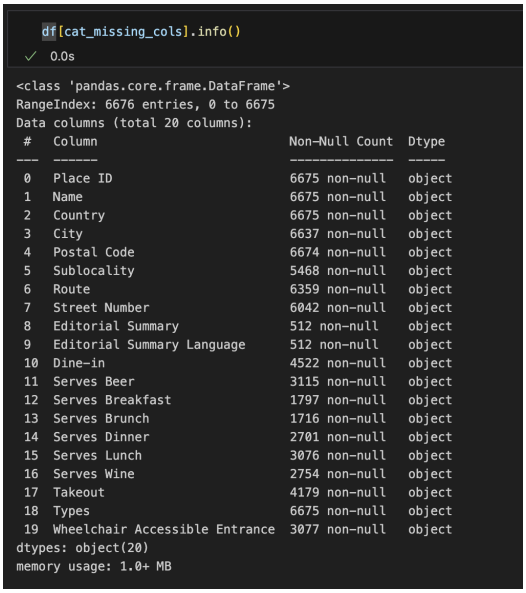


Figure 17. Summary of non-null values count for the dataset columns before missing values encoding.

```
df[cat_missing_cols].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6676 entries, 0 to 6675
Data columns (total 20 columns):
 #   Column                          Non-Null Count  Dtype 
---  ------                          --------------  ----- 
 0   Place ID                        6676 non-null   object
 1   Name                            6676 non-null   object
 2   Country                         6676 non-null   object
 3   City                            6676 non-null   object
 4   Postal Code                     6676 non-null   object
 5   Sublocality                     6676 non-null   object
 6   Route                           6676 non-null   object
 7   Street Number                   6676 non-null   object
 8   Editorial Summary               6676 non-null   object
 9   Editorial Summary Language      6676 non-null   object
 10  Dine-in                         6676 non-null   object
 11  Serves Beer                     6676 non-null   object
 12  Serves Breakfast                6676 non-null   object
 13  Serves Brunch                   6676 non-null   object
 14  Serves Dinner                   6676 non-null   object
 15  Serves Lunch                    6676 non-null   object
 16  Serves Wine                     6676 non-null   object
 17  Takeout                         6676 non-null   object
 18  Types                           6676 non-null   object
 19  Wheelchair Accessible Entrance  6676 non-null   object
dtypes: object(20)
memory usage: 1.0+ MB
```

Figure 18. Summary of non-null values count for the dataset columns after missing values encoding.

Previously, an example illustrated the encoding of missing values. Considering that missing data columns in the dataset could be estimated based on other columns (e.g., if a restaurant serves wine, it is likely to serve beer and dinner), and vice versa. Similarly, working hours could be approximated based on the type of establishment and the average working hours of similar places. Therefore, exploring data imputation on the dataset becomes a valuable consideration.

Imputation of Missing Values

One approach to address missing values is imputation, where estimates fill in the gaps based on information and relationships within the dataset. This involves leveraging patterns from attributes with non-missing values to provide reliable estimates for the missing ones.

One of the widely adopted techniques for imputing missing values is the K-Nearest Neighbors algorithm.

K-Nearest Neighbors (KNN) is a straightforward machine learning algorithm used for classification and regression tasks. In KNN, data points are represented as vectors in a multi-dimensional space, and predictions are made based on the majority class of the k-nearest neighbors for classification or the average of their values for regression. The algorithm calculates distances between the data point to be classified and all other data points, then selects the k-nearest neighbors. The final prediction is determined by majority voting for classification or averaging for regression among these neighbors.

However, a drawback of KNN lies in its reliance on Euclidean distance calculations, rendering it sensitive to outliers. Additionally, KNN struggles with categorical data, necessitating data transformation, and performs optimally when the data is scaled. These challenges are effectively addressed by leveraging Random Forest-based imputation methods.

MissForest algorithm for Missing Values Imputation

In this study, a more sophisticated approach is explored, the MissForest algorithm - implemented through the IterativeImputer from the scikit-learn machine learning library for Python, utilizing Random Forests as regressors to mimic its behaviour. MissForest is an imputation algorithm tailored to handle missing values, particularly in datasets featuring both numerical and categorical variables. The algorithm employs a random forest strategy to iteratively impute missing values.

The imputation process is iterative, refining imputations over multiple passes to continually enhance accuracy. Notably, MissForest introduces a stochastic element, inherent to the random forest strategy, adding randomness to the imputation process. This stochasticity contributes to the algorithm's resilience and effectiveness in handling missing data across diverse datasets.

Algorithm Overview:

1. Initialization:
   - Initialize the dataset, replacing all missing values with a placeholder (usually NaN).

2. Iterative Imputation:
   - Perform iterative imputation for a specified number of iterations or until convergence.
   - Build a random forest in each iteration using observed data for variables with missing values.
   - Impute missing values for each variable using predictions from the corresponding random forest.
   - Repeat the process for all variables with missing values.

3. Convergence:
   - Continue iterations until a stopping criterion is met. Convergence occurs when imputed values stabilize or show minimal changes.

4. Output:
   - Obtain a final dataset with imputed values for all missing entries.

As the RandomForestClassifier cannot handle categorical values directly, the categorical columns underwent encoding using scikit-learn's LabelEncoder. By employing scikit-learn's IterativeImputer with RandomForest as an estimator, the behaviour of the MissForest imputation algorithm was successfully replicated.

```python
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.preprocessing import LabelEncoder

# Identify numerical and categorical columns
num_cols = df.select_dtypes(include=[np.number]).columns
cat_cols = df.select_dtypes(include=[object]).columns

# Initialize a dictionary to hold the LabelEncoders for each column
le_dict = {}

for col in cat_cols:
    # Initialize LabelEncoder
    le = LabelEncoder()

    # Fit and transform the non-missing strings to numerical labels
    non_missing = df[col][df[col].notna()]
    df.loc[df[col].notna(), col] = le.fit_transform(non_missing)

    # Store the LabelEncoder for later use
    le_dict[col] = le

# Impute missing values for numerical columns
num_imp = IterativeImputer(RandomForestRegressor(n_estimators=10, max_depth=10), max_iter=10, random_state=0)
df[num_cols] = num_imp.fit_transform(df[num_cols])

# Impute missing values for categorical columns
cat_imp = IterativeImputer(RandomForestClassifier(n_estimators=10, max_depth=10), max_iter=10, random_state=0)
df[cat_cols] = np.round(cat_imp.fit_transform(df[cat_cols]))  # Round to nearest integer

# Convert the imputed numerical values back to categorical
for col in cat_cols:
    df[col] = le_dict[col].inverse_transform(df[col].astype(int))
```

Figure 19. Imputation of the missing dataset values using scikit-learn IterativeImputer.

By using Pandas' head method, the first five rows of the dataset can be compared before and after the imputation was applied.

```python
pd.set_option('display.max_columns', None)
df.head()
```

✓ 0.0s

	Place ID	Name	City	Postal Code	Sublocality	Route	Street Number	Latitude	Longitude	Rating	Reviews Count	Price Level	Dine-in	Monday_Open	Monday_Close	Tuesday_Open	Tuesday_Close	Wednesday_Open	Wednesday_Close
0	ChIJew3oD1vcQUcR88sk5uM2xYc	All About Street Food	Budapest	1088	VIII. kerület	Múzeum utca	7	47.490103	19.063370	3.9	144.0	2.0	True	1100.0	2245.0	1100.0	2245.0	1100.0	2245.0
1	ChIJJzNmdZLdQUcROh6kUWaeAr4	Street food	Budapest	1042	V. kerület	József Attila utca	31	47.499184	19.052779	4.7	22.0	NaN	True	NaN	NaN	NaN	NaN	NaN	NaN
2	ChIJew3oD1vcQUcR88sk5uM2xYc	All About Street Food	Budapest	1088	VIII. kerület	Múzeum utca	7	47.490103	19.063370	3.9	144.0	2.0	True	1100.0	2245.0	1100.0	2245.0	1100.0	2245.0
3	ChIJkd1d4mrcQUcR53ZWiBrMhtA	Meatology Budapest	Budapest	1065	VI. kerület	Bajcsy-Zsilinszky út	15/b	47.500832	19.055246	4.6	3252.0	2.0	True	1130.0	2200.0	1130.0	2200.0	1130.0	2200.0
4	ChIJJzNmdZLdQUcROh6kUWaeAr4	Street food	Budapest	1042	V. kerület	József Attila utca	31	47.499184	19.052779	4.7	22.0	NaN	True	NaN	NaN	NaN	NaN	NaN	NaN

Figure 20. The first 5 rows of the dataset, before the imputation of missing values, the first part.

Thursday_Open	Thursday_Close	Friday_Open	Friday_Close	Saturday_Open	Saturday_Close	Sunday_Open	Sunday_Close	Serves Beer	Serves Breakfast	Serves Brunch	Serves Dinner	Serves Lunch	Serves Wine	Takeout	Types	User Ratings Total	Wheelchair Accessible Entrance
1100.0	2255.0	1100.0	2245.0	1100.0	2245.0	1100.0	2245.0	True	NaN	False	True	True	True	True	meal_delivery, meal_takeaway, restaurant, food...	144.0	False
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	True	NaN	True	restaurant, food, point_of_interest, establish...	22.0	NaN
1100.0	2255.0	1100.0	2245.0	1100.0	2245.0	1100.0	2245.0	True	NaN	False	True	True	True	True	meal_delivery, meal_takeaway, restaurant, food...	144.0	False
1130.0	2200.0	1130.0	2200.0	1130.0	2200.0	1130.0	2200.0	True	True	True	True	True	True	True	restaurant, bar, food, point_of_interest, esta...	3252.0	True
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	True	NaN	True	restaurant, food, point_of_interest, establish...	22.0	NaN

Figure 21. The first 5 rows of the dataset, before the imputation of missing values, the second part.

After applying the IterativeImputer all of the missing values have been imputed. Notably, for the opening and closing times, the algorithm approximated the times based on places it deemed similar and considered the overall values within the dataset.

```
df.head()
```
✓ 0.0s Python

	Place ID	Name	City	Postal Code	Sublocality	Route	Street Number	Latitude	Longitude	Rating	Reviews Count	Price Level	Dine-in	Monday_Open	Monday_Close	Tuesday_Open	Tuesday_Close	Wednesday_Open	Wednesday_Close
0	ChIJew3oD1vcQUcR88sk5uM2xYc	All About Street Food	Budapest	1088	VIII. kerület	Múzeum utca	7	47.490103	19.063370	3.9	144.0	2.000000	True	1100.0000	2245.000000	1100.000000	2245.0	1100.000000	2245.000000
1	ChIJJzNmdZLdQUcROh6kUWaeAr4	Street food	Budapest	1042	V. kerület	József Attila utca	31	47.499184	19.052779	4.7	22.0	1.590734	True	1214.0625	1792.260299	1235.363636	1800.0	1227.912848	1800.314823
2	ChIJew3oD1vcQUcR88sk5uM2xYc	All About Street Food	Budapest	1088	VIII. kerület	Múzeum utca	7	47.490103	19.063370	3.9	144.0	2.000000	True	1100.0000	2245.000000	1100.000000	2245.0	1100.000000	2245.000000
3	ChIJkd1d4mrcQUcR53ZWlBrMhtA	Meatology Budapest	Budapest	1065	VI. kerület	Bajcsy-Zsilinszky út	15/b	47.500832	19.055246	4.6	3252.0	2.000000	True	1130.0000	2200.000000	1130.000000	2200.0	1130.000000	2200.000000
4	ChIJJzNmdZLdQUcROh6kUWaeAr4	Street food	Budapest	1042	V. kerület	József Attila utca	31	47.499184	19.052779	4.7	22.0	1.590734	True	1214.0625	1792.260299	1235.363636	1800.0	1227.912848	1800.314823

Figure 22. The first 5 rows of the dataset, after the imputation of missing values, the first part.

Python

Thursday_Open	Thursday_Close	Friday_Open	Friday_Close	Saturday_Open	Saturday_Close	Sunday_Open	Sunday_Close	Serves Beer	Serves Breakfast	Serves Brunch	Serves Dinner	Serves Lunch	Serves Wine	Takeout	Types	User Ratings Total	Wheelchair Accessible Entrance
1100.0	2255.00000	1100.00000	2245.000000	1100.000000	2245.000000	1100.000000	2245.000000	True	False	False	True	True	True	True	meal_delivery, meal_takeaway, restaurant, food...	144.0	False
1228.5	1802.97416	1208.56356	1804.374425	1364.239649	1582.640756	1325.653061	1652.290157	True	True	True	True	True	True	True	restaurant, food, point_of_interest, establish...	22.0	True
1100.0	2255.00000	1100.00000	2245.000000	1100.000000	2245.000000	1100.000000	2245.000000	True	False	False	True	True	True	True	meal_delivery, meal_takeaway, restaurant, food...	144.0	False
1130.0	2200.00000	1130.00000	2200.000000	1130.000000	2200.000000	1130.000000	2200.000000	True	True	True	True	True	True	True	restaurant, bar, food, point_of_interest, esta...	3252.0	True
1228.5	1802.97416	1208.56356	1804.374425	1364.239649	1582.640756	1325.653061	1652.290157	True	True	True	True	True	True	True	restaurant, food, point_of_interest, establish...	22.0	True

Figure 23. The first 5 rows of the dataset, after the imputation of missing values, the second part.

Feature Engineering

Feature engineering is the process of creating new features or modifying existing ones in a dataset to enhance the performance of machine learning models. It involves transforming raw data into a more suitable format, extracting relevant information, and generating additional features that can contribute to the model's predictive power.

For the dataset, feature engineering is necessary for the "Types" column. Within "Types" each entry contains a list of types relevant to the place. The column needs transformation into a one-hot encoded format for easier processing by machine learning algorithms.

The lists were split, and the pandas value counts function was applied to obtain the count of each type appearing within the column.

```
df['Types'] = df['Types'].str.split(', ')
# create a new dataframe with the counts of unique values in the 'types' column
counts = pd.DataFrame(df['Types'].explode().value_counts())

# print the counts
print(counts)
print(len(counts))
```

✓ 0.0s

Types	count
point_of_interest	6606
establishment	6606
food	5770
restaurant	3623
store	2128
bar	783
grocery_or_supermarket	721
cafe	690
bakery	399
supermarket	383
liquor_store	187
meal_delivery	157
meal_takeaway	100
convenience_store	82
lodging	81
home_goods_store	59
night_club	57
health	51
travel_agency	27
tourist_attraction	25
gas_station	24
shopping_mall	21
furniture_store	21
clothing_store	21
electronics_store	19
drugstore	15
department_store	14
hardware_store	14
park	10
spa	7
school	7
car_wash	7
museum	5
book_store	5
florist	4
general_contractor	4
bowling_alley	4
art_gallery	4
transit_station	3
parking	3
university	3
storage	3
finance	2
gym	2
atm	2
jewelry_store	2
car_repair	2
movie_theater	1
subway_station	1
zoo	1
car_dealer	1
car_rental	1
amusement_park	1
casino	1
beauty_salon	1
train_station	1

56

Figure 24. Value counts of each type of the “Types” column.

The "Type" column was converted into 56 one-hot-encoded columns using the pandas `get_dummies` function.

```
# Convert the types into one-hot encoding
type_df = pd.get_dummies(df['Types'].apply(pd.Series).stack()).groupby(level=0).sum()

# Add a prefix to the column names
type_df = type_df.add_prefix('Type_')

# Join the one-hot encoded columns to the original dataframe
df = pd.concat([df, type_df], axis=1)

# Drop the original 'Types' column
df = df.drop('Types', axis=1)
```

✓ 0.7s Python

```
df.head(5)
```

✓ 0.0s Python

Type_amusement_park	Type_art_gallery	Type_atm	Type_bakery	Type_bar	Type_beauty_salon	Type_book_store	Type_bowling_alley	Type_cafe	Type_car_dealer	Type_car_rental	Type_car_repair	Type_car_wash	Type_casino	Type_clothing_store	Type_
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 25. Conversion of the “Types” column into one-hot encoded columns.

Some of the columns and entries that somehow entered the dataset are not relevant for our application, the example are entries with “Type_hardware_store” column equal to 1, those entries could accidentally get into the dataset during data collection phase and need to be dropped.

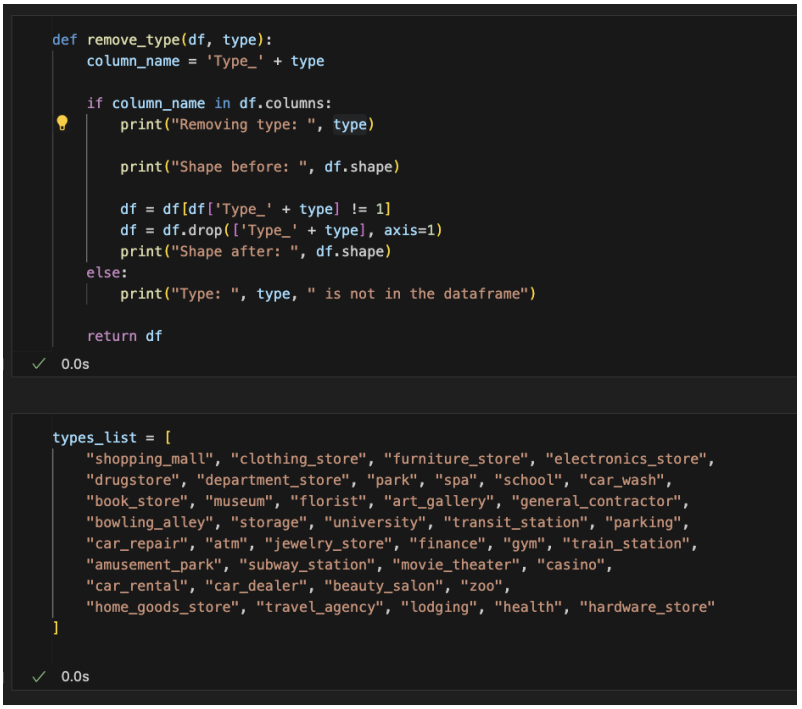
```
# Find rows where 'Type_hardware_store' is true
hardware_store_rows = df[df['Type_hardware_store'] == 1]

# Print these rows
hardware_store_rows.head(15)
```

✓ 0.0s Python

	Place ID	Name	City	Postal Code	Sublocality	Route	Street Number	Latitude	Longitude	Rating	Reviews Count	Price Level	Dine-in	Monday_Open	Monday_Close	Tuesday_Open	Tuesday_Close	Wednesday_Open	Wednes
288	ChIJ_91q5EHZQUcRQIf1h02hAgM	Praktiker	Budapest	1034	III. kerület	Flórián tér	1	47.541770	19.028634	4.1	5628.0	1.771534	True	800.0	2000.0	800.0	2000.0	800.0	
883	ChIJ-zPk7SPaQUcRURL7o7keigg	Praktiker	Budapest	1044	IV. kerület	Váci út	60-62	47.571126	19.079133	4.1	4439.0	1.950193	True	800.0	2000.0	800.0	2000.0	800.0	
946	ChIJNwGXu87BQUcRDytnKWIjsG0	Praktiker	Vecsés	2220	Érdliget	Fő út	246-248	47.417405	19.249256	4.1	4110.0	1.962357	True	800.0	2000.0	800.0	2000.0	800.0	
1766	ChIJlaDr2hrdQUcRWB81RA5OKWw	OBI áruház Budapest	Budapest	1095	IX. kerület	Soroksári út	33	47.466970	19.075443	4.3	9797.0	1.807841	True	800.0	2000.0	800.0	2000.0	800.0	
1967	ChIJRXI2qd7pQUcRXlCc7I3Xtdo	OBI áruház Budapest, Soroksár	Budapest	1239	XXIII. kerület	Bevásárló utca	6	47.413873	19.161754	4.3	6117.0	1.821010	True	800.0	2000.0	800.0	2000.0	800.0	
2008	ChIJle8WGoHnQUcRvRG9gt9HDf4	OBI áruház Budapest, Savoya Park	Budapest	1117	XI. kerület	Hunyadi János út	23	47.436814	19.041864	4.3	5550.0	2.058547	True	800.0	2000.0	800.0	2000.0	800.0	
2359	ChIJaQItdCTeQUcRvU5zJkpq5Jw	OBI	Budapest	1191	XIX. kerület	Vak Bottyán utca	75/a	47.462838	19.144455	4.3	4602.0	1.962357	True	800.0	2000.0	800.0	2000.0	800.0	
2998	ChIJDbVo2OHCQUcRjKm6ueS1Lqg	OBI áruház Budapest	Budapest	1046	IV. kerület	Szent Imre utca	1	47.568944	19.097175	4.2	5392.0	1.950193	True	800.0	2000.0	800.0	2000.0	800.0	
3181	ChIJqWiGMTfdQUcRDFMbDlVZYpI	Praktiker	Budapest	1095	IX. kerület	Mester utca	87	47.469882	19.082120	4.1	7332.0	1.807841	True	800.0	2000.0	800.0	2000.0	800.0	
3226	ChIJdVTxhLz3aUcR70JKDM3pkpI	OBI	Székesfehérvár	8000	Agárd	Budai út	41	47.190739	18.423797	4.3	4173.0	2.058547	True	800.0	2000.0	800.0	2000.0	800.0	
3502	ChIJV67KP2_XQUcReXP_X0M7Xh4	OBI Budakalász	Budakalász	2011	XXII. kerület	Omszk park	2	47.610978	19.060039	4.4	2986.0	1.912923	True	800.0	2000.0	800.0	2000.0	800.0	
3806	ChIJOUSyoR3eQUcRDB5I-MaBqeE	Praktiker	Budaörs	2040	Budaörsi Kamaraerdő	Malomkő utca	3	47.452836	18.971488	4.1	4859.0	2.058547	True	800.0	2000.0	800.0	2000.0	800.0	
5989	ChIJKXcvn8tjakcR1A-5kNugUb0	Praktiker	Esztergom	2500	Agárd	Dobogókői út	41	47.773835	18.752553	4.0	2313.0	2.002693	True	800.0	2000.0	800.0	2000.0	800.0	
6401	ChIJs_8zvZ7aQUcR3gHwnmWPfrk	OBI áruház Fót	Fót	2151	Budaörsi Kamaraerdő	Fehérkő utca	2	47.578070	19.161060	4.5	8003.0	1.921010	True	800.0	2000.0	800.0	2000.0	800.0	

Figure 26. Dataset entries with the “Type_hardware_store” column equal to 1.

Automating the process of removing irrelevant columns and entries can be achieved by developing a function that identifies and drops data entries based on provided one-hot column values.



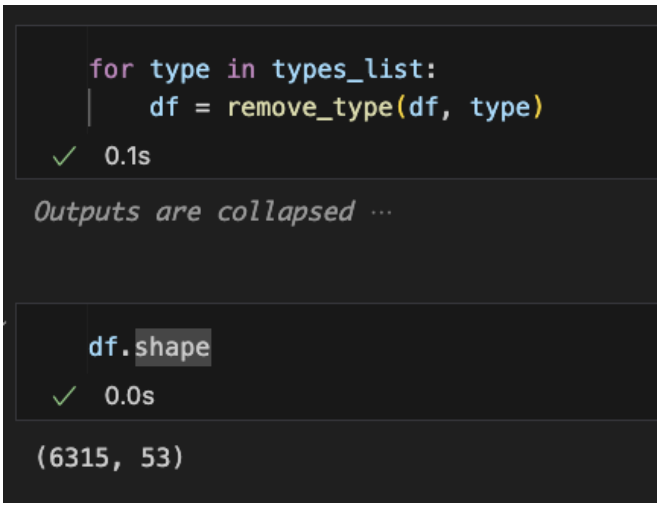
```
def remove_type(df, type):
    column_name = 'Type_' + type

    if column_name in df.columns:
        print("Removing type: ", type)

        print("Shape before: ", df.shape)

        df = df[df['Type_' + type] != 1]
        df = df.drop(['Type_' + type], axis=1)
        print("Shape after: ", df.shape)
    else:
        print("Type: ", type, " is not in the dataframe")

    return df
```

✓ 0.0s

```
types_list = [
    "shopping_mall", "clothing_store", "furniture_store", "electronics_store",
    "drugstore", "department_store", "park", "spa", "school", "car_wash",
    "book_store", "museum", "florist", "art_gallery", "general_contractor",
    "bowling_alley", "storage", "university", "transit_station", "parking",
    "car_repair", "atm", "jewelry_store", "finance", "gym", "train_station",
    "amusement_park", "subway_station", "movie_theater", "casino",
    "car_rental", "car_dealer", "beauty_salon", "zoo",
    "home_goods_store", "travel_agency", "lodging", "health", "hardware_store"
]
```

✓ 0.0s

Figure 27. A function for automatic removal of data entries and column dropping based on the provided column name, the list of columns that are irrelevant to our application.



```
for type in types_list:
    df = remove_type(df, type)
```

✓ 0.1s

Outputs are collapsed ···

```
df.shape
```

✓ 0.0s

```
(6315, 53)
```

Figure 28. remove_type function run for all of the irrelevant types, the data frame shape after the run.

After running the remove_type function for all of the irrelevant types the dataset has been removed from 291 and 39 irrelevant rows and columns respectively.

4. Classification, Types of Classifiers

In a 2006 study titled "An Empirical Comparison of Supervised Learning Algorithms," the findings indicate that when considering eight different metrics, the most robust models are calibrated boosted trees, calibrated random forests, bagged trees, and neural networks. When calibration is not applied, the top-performing models overall include bagged trees, random forests, and neural networks. These results suggest that calibration, particularly in boosted trees and random forests, enhances performance across various evaluation metrics.
In a detailed comparison of various classification algorithms in 2006, it was observed that optimized decision tree methods specifically boosted trees or random forests, and surpassed other techniques such as Bayesian classification in terms of performance [8].

4.1 Decision trees

Decision tree learning is a popular supervised learning method employed in statistics, data mining, and machine learning. It utilizes classification or regression decision trees as predictive models to draw conclusions about a set of observations.

These decision trees are valued for their simplicity and interpretability, making them among the most favored machine learning algorithms. In decision analysis, they serve as explicit representations of decisions and decision-making processes. In data mining, a decision tree acts descriptively, often becoming an input for subsequent decision-making steps.

Assuming finite discrete domains for input features and a single target feature called "classification," each internal node in a classification decision tree is labeled with an input feature. The arcs from this node are labeled with possible values of the target feature, leading either to a subordinate decision node or a leaf node representing a class or a probability distribution over classes.
The process of constructing a decision tree involves recursively splitting the source set, starting from the root node. Using a greedy algorithm, the process continues until subsets have uniform values or further splitting doesn't improve predictions. [9][10].

This recursive partitioning process, guided by splitting rules and classification features, defines the construction of decision trees [11]. It offers a versatile approach for various predictive modeling tasks, making it a valuable tool in the field of machine learning.

Two main types of decision trees are prevalent in data mining:

1. Classification Tree Analysis: Predicts the discrete class to which the data belongs.
2. Regression Tree Analysis: Predicts a real number, such as the price of a house or a patient's length of stay in a hospital.

4.2 Boosted trees

In the realm of machine learning, boosting is an ensemble meta-algorithm designed primarily to diminish bias and, to some extent, variance in supervised learning [12]. It belongs to a family of machine learning algorithms that elevate the performance of weak learners to that of strong ones [13].

The process of ensemble building entails incrementally training each new instance with a focus on the training instances previously mis-modeled. One specific algorithm within the boosted trees category is AdaBoost. In AdaBoost, multiple "weak learners," which are decision trees with just one level of depth (referred to as decision stumps), are employed. The algorithm trains each decision stump while emphasizing data samples that were incorrectly classified by the previous stump. This emphasis is achieved by assigning more weight to the misclassified samples during the training process.

4.3 Bagged trees

Bootstrap aggregated, commonly known as bagged decision trees, is an early ensemble method. It constructs multiple decision trees by repeatedly resampling training data with replacement (bootstrap sampling), building individual trees on each of these samples, and then combining the predictions of these trees through voting to arrive at a consensus prediction [14].
The process helps improve the overall performance and robustness of the model by reducing overfitting and variance. The aggregated trees function collaboratively, with each tree concentrating on a distinct perspective of the data. The combination of their individual decision-making processes contributes to enhanced accuracy and stability in predictions.
A random forest classifier is a particular instance of bootstrap aggregating.

4.4 Random forests

Random forests, also known as random decision forests, are an ensemble learning method used for classification, regression, and various other tasks. This technique constructs numerous decision trees during training. In classification tasks, the random forest outputs the class most frequently selected by the individual trees. In regression tasks, it returns the mean or average prediction from the individual trees [15][16]. The purpose of random decision forests is to address the tendency of decision trees to overfit their training set[17].

While random forests typically outperform standalone decision trees, their accuracy is generally lower than that of gradient boosted trees. It's important to note that the performance of random forests can be influenced by the characteristics of the data [18][19].

4.5 Artificial Neural Networks

Artificial neural networks (ANNs), also known as neural networks (NNs) or neural nets, are a category of machine learning models inspired by the organizational principles found in biological neural networks within animal brains [20][21].

An ANN is composed of interconnected units or nodes referred to as artificial neurons, which loosely mimic the neurons in biological brains. Similar to the synapses in a biological brain, each connection can transmit signals to other neurons. Artificial neurons receive, process, and transmit signals to connected neurons. The signals, represented by real numbers, undergo computation through a non-linear function of the sum of inputs. These connections are termed edges, and both neurons and edges typically possess weights that adjust during the learning process. These weights modulate the strength of signals, either increasing or decreasing it. Neurons may also incorporate a threshold, allowing a signal to be transmitted only if the cumulative signal surpasses this threshold.

Neurons are commonly organized into layers, where each layer may perform distinct transformations on its inputs. Signals traverse from the initial layer (input layer) to the final layer (output layer), possibly traversing these layers multiple times.

The No Free Lunch Theorem asserts that no one-size-fits-all learning algorithm universally outperforms others. Even top-performing models like calibrated boosted trees, random forests, and bagged trees may exhibit poor performance on certain problems or metrics. Conversely, models with generally low average performance might excel in specific situations.

Given this variability in performance across different problems, one strategy to enhance classifier performance is to create a hybrid classifier. This involves combining multiple classifiers to tailor the system to the specific characteristics of the problem.

This paper tested the classifier workflow using scikit-learn's RandomForest classifier. Initially, a "User_Preference" column was established to indicate user preferences for each restaurant: 0 denotes that the user does not like or has an unknown preference, while 1 indicates that the user likes the restaurant.

Figure 29. Creation of the “User_Preference” column filled with zeros.

Next, the places liked by users were identified, and their corresponding values in the "User_Preference" column were set to 1.

Figure 30. Setting “User_Preference” column value to 1 of the restaurant samples user likes.

Given the limited data on user preferences, the occurrences of likes in the dataset are significantly fewer than the instances of dislikes or unknown preferences. Consequently, the samples liked by users are extracted for exclusive use in the training split of the dataset.

Figure 31. Take out of the liked by the user samples for later use of them solely in the training split of the dataset.

Next, the data is divided into training and testing sets for our algorithms. Unnecessary columns such as "Name" and "Place ID" are dropped from the training data, and label columns are separated accordingly.

```
from sklearn.model_selection import train_test_split
columns_to_drop = ['Name', 'Place ID', 'User_Preference']
X = df_without_place_ids_for_train_df.drop(columns_to_drop, axis=1)
y = df_without_place_ids_for_train_df['User_Preference']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
✓ 0.0s
```

Figure 32. Split of the dataset into train and test data.

This is a typical representation of the training data for our dataset, where X represents input features and y represents labels. It's important to note that due to the cold-start problem, characterized by limited information on user preferences, a small subset of the training data needs to be used. This ensures that the algorithm doesn't converge to the simplest solution of outputting only zeros, which dominate the dataset.

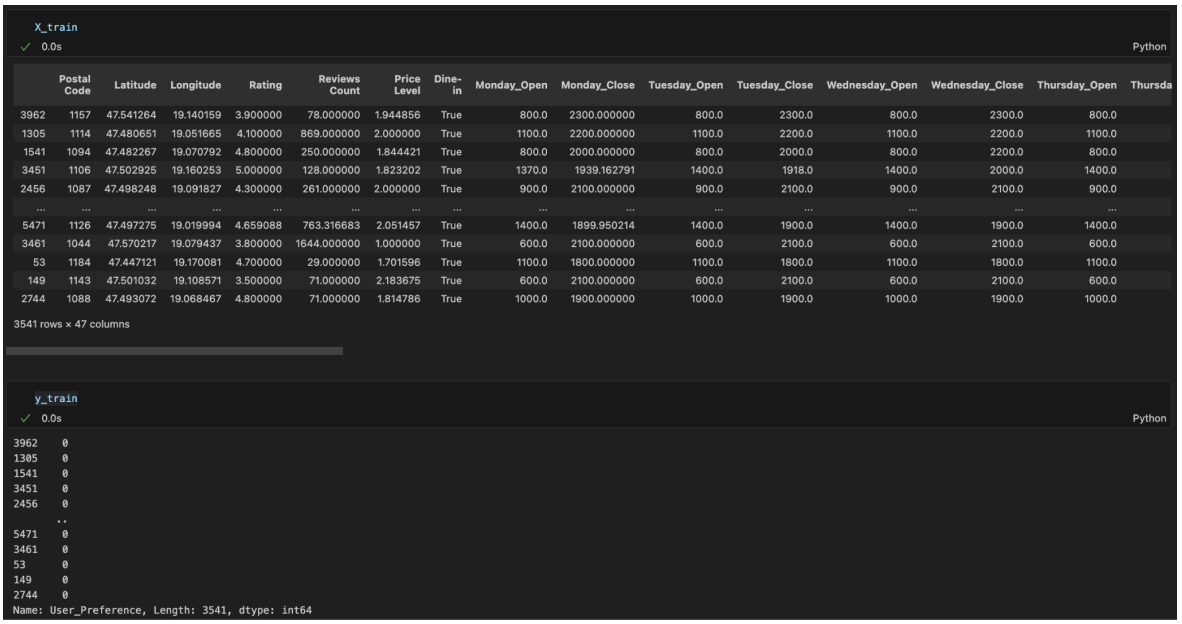


Figure 33. Dataset training data.

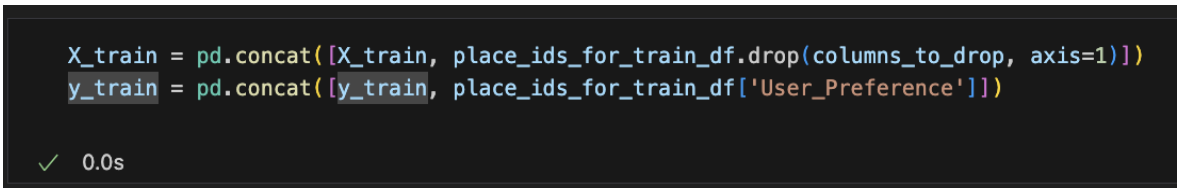


Figure 34. Appending of the liked by the user samples to the training split of the dataset.

Following that, a classifier is trained, with RandomForestClassifier being the chosen model in this instance.

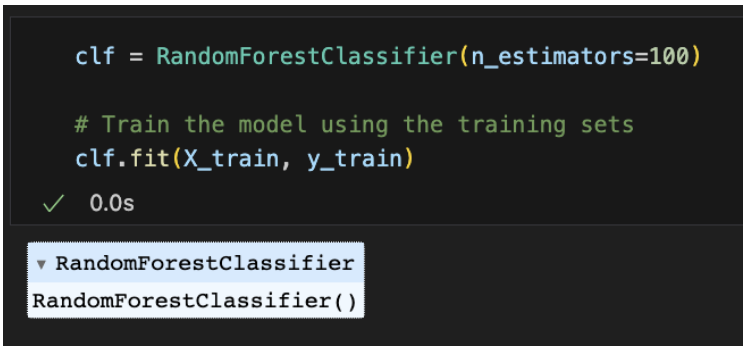


Figure 35. RandomForestClassifier training.

Upon completing the classifier training, its application to predict the test subset of the data involves applying a mask to filter and extract only positive predictions. In this context, these positive predictions correspond to restaurant recommendations.

```
pred_all = clf.predict(df.drop(columns_to_drop, axis=1))
```

0.0s Python

```
# Get boolean mask where y_pred is 1
mask = (pred_all == 1)

# Use this mask to filter X_test
selected_rows = df[mask]

# Print the selected rows
selected_rows[10:50]
```

0.0s Python

	Place ID	Name	Postal Code	Latitude	Longitude	Rating	Reviews Count	Price Level	Dine-in	Monday_Open	Monday_Close	Tuesday_Open	Tuesday_Close	Wednesday_Open	We
365	ChIJDXQmoRjdQUcRLkIEOqprQZk	Ultramarinos Martinez spanyol DELI	1111	47.481728	19.052791	5.0	187.0	1.920385	True	1700.000000	2300.000000	1700.0	2300.0	1700.0	
416	ChIJqwOMzmrcQUcR4YaUUe7zMV4	California Coffee Company	1051	47.500384	19.052974	3.9	2187.0	2.000000	True	700.000000	2300.000000	700.0	2300.0	700.0	
460	ChIJQW-I3_rdQUcRVBYOwTfnEek	Asian Street Food Budapest	1076	47.502133	19.084639	4.7	1203.0	1.919771	True	1100.870575	2099.878049	1100.0	2100.0	1100.0	
471	ChIJQYNiDFDdQUcRLpO7iKagjnE	Fleischer Restobar	1065	47.502859	19.061451	4.8	384.0	2.001722	True	1200.000000	2330.000000	1200.0	2330.0	1200.0	
475	ChIJsd91vffcQUcR-NefKyUZhEA	Starbucks Corvin	1083	47.486349	19.074309	4.2	1164.0	3.000000	True	700.000000	2100.000000	700.0	2100.0	700.0	

Figure 36. Predictions of the trained classifier.

5. Incorporating textual data input into recommender systems

In the realm of recommender systems, various challenges are encountered. When it comes to recommending places, the primary problem lies in obtaining detailed information about these locations. Typically, the available information in a standard format, such as restaurant names, addresses, opening hours, price levels, and average ratings, can be accessed without complex preprocessing and is publicly accessible. However, relying solely on this general data is insufficient for creating personalized and specific recommendations. The valuable insights often reside in the textual content of reviews and descriptions.

To address this issue, this section explores the application of one of the approaches for extracting meaningful insights from natural language. These insights can be instrumental in enhancing recommender systems, enabling them to provide more tailored and accurate recommendations based on the rich information found within textual reviews and descriptions.

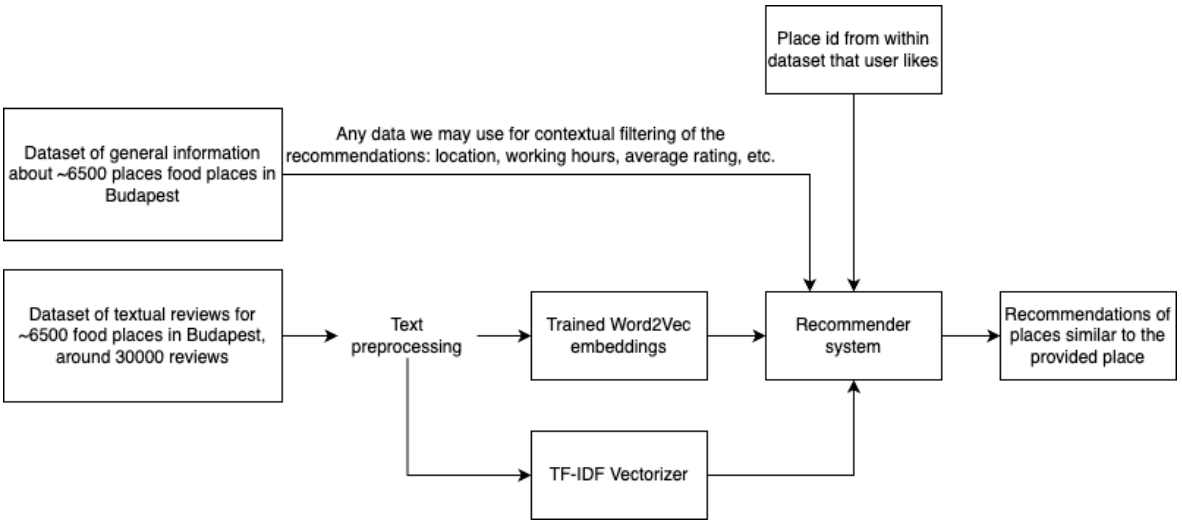


Figure 37. Recommender based on textual reviews logic design.

5.1 Word2Vec

Word2Vec is a group of Natural language processing model architectures for learning word embeddings from large datasets created and popularized by researchers at Google.

Word2Vec is a powerful tool for generating high-quality word vectors from massive datasets containing billions of words and millions of unique words in vocabulary. Unlike previous architectures, which struggled to handle more than a few hundred million words and typically produced word vectors with dimensions ranging from 50 to 100, Word2Vec excels in training on extensive datasets, providing an opportunity to capture intricate word relationships and meanings [22][23].

Word2Vec learns by looking at the context in which words appear in a large dataset. It assumes that words with similar meanings tend to appear in similar contexts.
Each word is represented as a vector (a series of numbers) and the position of a word's vector in this multi-dimensional space reflects its relationships with other words. Words that often appear together will have vectors that point in similar direction, the distance and direction between word vectors signify the relationships between the words.
Word2Vec uses a neural network to train the model on a large corpus of text. The model adjusts the word vectors during training to accurately predict the words that are likely to appear in a given context.

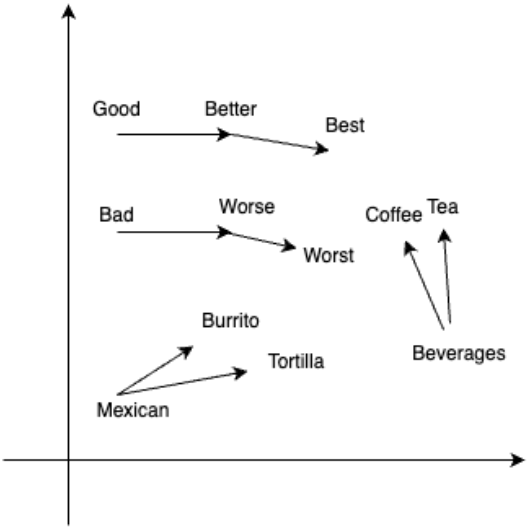


Figure 38. Word2Vec simplified vectors visualization example.

5.1.1 Continuous Bag-of-Words Model

The continuous bag-of-words model predicts the central word using nearby words in the context, including a few words before and after the target word. In this architecture, the order of words in the context doesn't matter, which is why it's called a bag-of-words model [22].

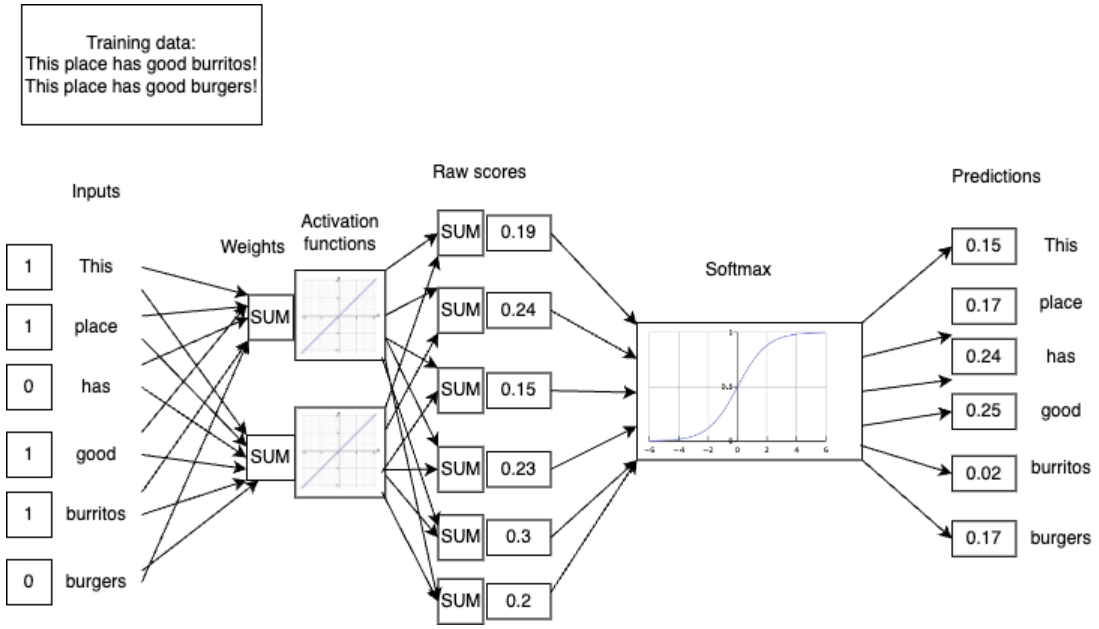


Figure 39. Word2Vec Continuous Bag-of-Words example, randomly initialized weights.

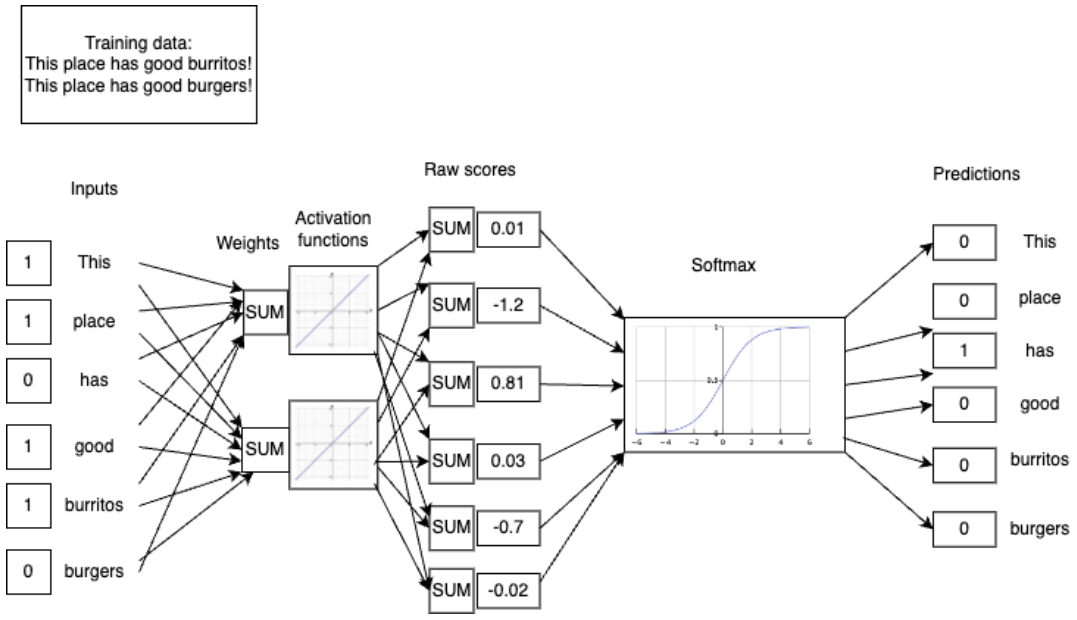


Figure 40. Word2Vec Continuous Bag-of-Words example, trained.

5.1.2 Continuous Skip-gram Model.

Skip-gram models continuously predict words within a specified range before and after the current word in a sentence. [22]

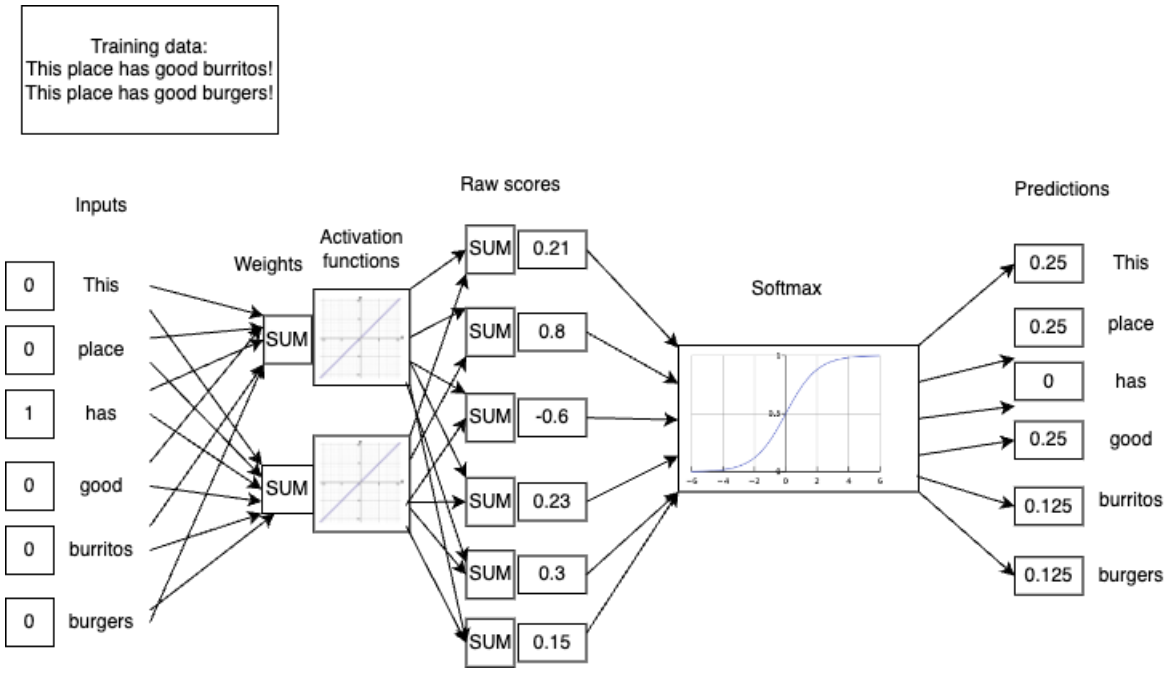


Figure 41. Word2Vec Skip-Gram example, trained.

5.2 Dataset

For the application of textual data into our recommender system the dataset used in this research comprises names and reviews of around 6500 establishments located in Budapest. To streamline the analysis, the top 5 pertinent reviews for each place were condensed into a single text. Each row in the dataset corresponds to a specific establishment and includes unique identifiers (place_id), the establishment's name, and the combined text of its top 5 relevant reviews.

place_ids	name	reviews_text
ChIJ0WX1KxDdQUcRt6sW8FXdbQM	Blue Agori Madach Greek Street Food Bar	The worst pork souvlaki platter i have ever tasted Full of fats, not even one inch of meat I spent my time chewing it in order to swallow Even taste was horrible Not the souvlaki spices Ok food,but does not beat greek food that we have tried before. It was relati Not as in Greece, but for Budapest a very good one you can get. Well done!
ChIJ1ek1d2jcQUcR1wtOR6mhePU	Falafel Bar	The staff is so warm and welcoming, excellent service! And the food is fantastic, best hummus I've e So fresh and colorful food too! Really recommend this place! We stumbled upon it as we were explo
ChIJ2eWPmRXdQUcRRFGehEBb9vM	MINO'S	Great service, the owner is really kind, and the food is soooo delicious!! If you are craving some auth
ChIJ3TKTdjbXQUcRzI2AD-cjL3k	Corner Street Food	Decent quality, good range, okay prices. What's not to like? Brilliant menu, excellent cook, delicious
ChIJ5959lZDdQUcRlkz4FBUgHIM	Papitos Mexican Street Food - Madách	5/5 reviews were correct! Deff recommend. Food. Burrito was great. I had it with pork, however with beef is better (tastier). Also ordered 3 tacos. Pork, chicken and beef. Supreme taste of beef and chicken. Pork is also good Both dishes come with tortilla chips. Ordered a medium hot sauce which was perfect for the dish i h Service. The guys who take orders and serve were nice and their suggestions were highly noted and conside Atmosphere Small cosy place, has space also outside (2 tables). Tacos are edible but nothing more, dry. They we Great vibe, great burrito, great beer. The pork burrito is really good, with fresh guac, and comes with 5/5.
ChIJ7U0lrUTcQUcRMEO3WEX1-SY	Vega City	I really liked the concept of the establishment where you can choose what kind of food you want an I would like more raw items and different freshes. Interesting vegan restaurant, canteen style with a Mostly enjoyed the atmosphere and the concept. Perfect if you want to get your food quickly and d Had a very good spicy ginger beer as well. They offer a daily menu in normal or large. You get an ou I took the daily 'Big' meal (Monday) and it was a good portion for my active American appetite. Chin
ChIJ8z64GGjcQUcRDtjy8jgvJIo	Street Food Karavan Budapest	A true gem we stumbled across almost randomly. Great variety of street food, from local specialties Couple that with great atmosphere and life-saving sprinklers cooling down the air, and you've got yo A great atmosphere with lots to eat and drink, cheap and some of the best renditions of authentic H The langos, goulash as well as the Mexican food were all great! A must visit if you are in the city. Such a great concept, a load of food vans in one place providing a

Figure 42. An example of the establishment reviews dataset entries.

5.3 Text Preprocessing

Each entry of the dataset underwent the following text preprocessing steps.

Standardization

Text standardization is a process of removing punctuation, special characters, stop words, HTML elements, standardizing the word casing, and following any other steps for simplifying and standardizing the text.

Tokenization

Tokenization is a process of breaking down a text into individual units, such as words or phrases (tokens).

Vectorization

Vectorization is the process of converting textual data in the form of tokens into numerical vectors or arrays of numbers.

```
#Utitlity functions for removing ASCII characters, converting lower case, removing stop words, html and punctuation from description

def _removeNonAscii(s):
    return "".join(i for i in s if  ord(i)<128)

def make_lower_case(text):
    return text.lower()

def remove_stop_words(text):
    text = text.split()
    stops = set(stopwords.words("english"))
    text = [w for w in text if not w in stops]
    text = " ".join(text)
    return text

def remove_html(text):
    html_pattern = re.compile('<.*?>')
    return html_pattern.sub(r'', text)

def remove_punctuation(text):
    tokenizer = RegexpTokenizer(r'\w+')
    text = tokenizer.tokenize(text)
    text = " ".join(text)
    return text

grouped_reviews_df['cleaned'] = grouped_reviews_df['reviews_text'].apply(_removeNonAscii)

grouped_reviews_df['cleaned'] = grouped_reviews_df.cleaned.apply(func = make_lower_case)
grouped_reviews_df['cleaned'] = grouped_reviews_df.cleaned.apply(func = remove_stop_words)
grouped_reviews_df['cleaned'] = grouped_reviews_df.cleaned.apply(func=remove_punctuation)
grouped_reviews_df['cleaned'] = grouped_reviews_df.cleaned.apply(func=remove_html)
```

Figure 43. Dataset text preprocessing Python functions.

The text underwent preprocessing, which involved removing stop words, punctuation, potential HTML tags, and non-ASCII characters. Furthermore, all words were converted to lowercase. Upon reviewing the dataset post-preprocessing, it is evident that all the intended steps were successfully applied.

place_ids	name	reviews_text	cleaned
ChIJ-8bW...	Montázs A...	Best latte I have had in a while. The service is friendl...	best latte while service friendly comfortable spacious nice coffee date play da...
ChIJ-8mN...	VELVET cafe	Another great find in Budapest that was suggested t...	another great find budapest suggested us previous tenant rental apartment t...
ChIJ-952b...	Ibolya Espr...	Great bar, love the retro/vintage look. Quick service ...	great bar love retro vintage look quick service ice old beer great local spot up...
ChIJ-99Rb...	Guri Serhá...	After a recent renovation, the place lost it's atmosph...	recent renovation place lost atmosphere beers good always food offers gone ...
ChIJ-Q2D...	Sajtangyal	The raw milk is fantastic and the kefir is pretty good....	raw milk fantastic kefir pretty good would recommend cream though great tas...
ChIJ-Q6ns...	LEVES.	The sandwich is cheap but was just simply bad. All t...	sandwich cheap simply bad ingredients feel like frozen unfrozen day making b...
ChIJ-Q_G...	Bohos Far...	Top 😀👍	top
ChIJ-R4Fo...	Kávéház Kft.	Our tour guide - Maria Gabriella suggested us to co...	tour guide maria gabriella suggested us come coffee desserts gave special m...
ChIJ-SzKC...	Aus Kebab	Best chicken my wife has ever tasted. She demande...	best chicken wife ever tasted demanded include review seriously though keba...
ChIJ-TCiE...	Winners S...	Great sports pub for watching football. The pub bar ...	great sports pub watching football pub bar screens everywhere large screen ...
ChIJ-TMzI...	Lidl	Lidl stands out as one of the top, well-organized, an...	lidl stands one top well organized varied branches strongly endorse it never p...
ChIJ-UOW...	Briós	Everything was nice and tasty, you have to wait abou...	everything nice tasty wait 10 15 minutes seated weekends would recommend ...
ChIJ-UQlN...	Orient Res...	soup was a bit strange and oily, but other things not ...	soup bit strange oily things bad wow hidden gem place legit chinese food che...
ChIJ-UU5...	Fekete	Nice courtyard loved the atmosphere, nice selection...	nice courtyard loved atmosphere nice selection brunch dishes eggs fita chees...
ChIJ-V436...	Vietnami ...	Such a good and fulfilling Bahn Mi sandwich for suc...	good fulfilling bahn mi sandwich good price perfect take away park nearby lak...
ChIJ-Vf12j...	Matcha Ts...	I had a wonderful experience at Matcha Tsuki Tea H...	wonderful experience matcha tsuki tea house staff incredibly friendly welcomi...
ChIJ-WA9f...	Osmani Dö...	Update: I got the gyro bowl with fries. It tasted great...	update got gyro bowl fries tasted great quick cheap guy working spoke englis...
ChIJ-WXA...	UNCENSO...	A truly unique concept, combining an audio visual ex...	truly unique concept combining audio visual experience tasty 6 course meal 6...
ChIJ-WnA...	Pizza Mág...	Very good tasty pizza and ice cream. The place is q...	good tasty pizza ice cream place quite set outside kind people wonderful staf...
ChIJ-WxD...	Devil's Pré...	A little street food restaurant, but it worths the price...	little street food restaurant worths price sandwiches good portions also big st...
ChIJ-X0Fj...	Araz Resta...	I was recommended this restaurant as a great place ...	recommended restaurant great place daily menu exactly was large dining roo...

Figure 44. The result of running the preprocessing function on the dataset.

A word corpus is created by extracting words from the cleaned reviews dataset.

```
corpus = []
for words in grouped_reviews_df['cleaned']:
    corpus.append(words.split())
```

Figure 45. Code for building the corpus of the textual reviews.

The resulting corpus comprises unique words present in the dataset.

	0	1	2	3	4	5	6	7	8	9	10
0	strange	reason	enough	cashiers	prepared	stand	line	otherwise	would	give	5
1	sommelier	nicki	gave	us	great	education	hungarian	wine	especially	novice	really
2	best	budget	friendly	place	hangout	loved	ones	spent	city	vibe	them
3	tried	duck	liver	pate	breakfast	absolutely	amazing	served	couple	slices	milk
4	like	aldi	hungary	good	quality	products	good	prices	nice	always	find
5	avoid	avoid	avoid	place	take	prize	worst	bar	i	ve	ever
6	spar	partner	branch	products	higher	price	normal	spar	still	good	selection
7	750ft	os	hot	dog	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	like	market	budapest	one	pretty	new	renovated	find	lots	fresh	vegetables
9	open	business	hours	shown	page	first	time	either	even	advertised	facebook
10	stop	by	friend	suggested	unbelievable	selection	rear	wines	one	best	malbec
11	enjoyed	brunch	here	saturday	quite	crowded	inside	seating	ate	delicious	feta
12	nice	pleasant	little	find	bespoke	unique	like	spirit	s	love	cosy
13	good	specialty	coffee	place	many	budapest	tried	brasilian	coffee	had	cappuccino
14	1	spoiled	boba	black	crystal	boba	spoiled	despite	confirming	staff	acknowled...
15	delicious	chinese	food	ever	had	seriously	recommend	fried	rice	peppers	prawn
16	one	supermark...	people	actually	kind	always	clean	tidy	NaN	NaN	NaN
17	best	latte	while	service	friendly	comfortable	spacious	nice	coffee	date	play
18	another	great	find	budapest	suggested	us	previous	tenant	rental	apartment	there
19	great	bar	love	retro	vintage	look	quick	service	ice	old	beer
20	recent	renovation	place	lost	atmosphere	beers	good	always	food	offers	gone

Figure 46. A part of the resulting corpus.

5.4 Application of Word2Vec for Creating Recommendations

This paper will use pre-trained word embeddings from Google, which were trained on a vast corpus, including sources like Wikipedia and news articles. These pre-trained embeddings provide vectors for words, forming a comprehensive collection of word representations. Each word in the vocabulary corresponds to a specific vector.

In the context of converting reviews of establishments into vectors and determining the similarity between these vectors to make recommendations, the challenge is to transform each review into a sequence of words. Word2Vec provides vector representations for individual words. Two techniques are employed to generate recommendations: averaging Word2Vec vector embeddings and TF-IDF Word2Vec. These methods allow the transformation of entire reviews into meaningful vectors and assess their similarity for practical establishment recommendations.

5.4.1 Averaging Word2Vec vectors

Word2Vec assigns a D-dimensional vector to each word in a sentence. In our example, the description contains 23 words, denoted as w1, w2, w3, ..., w23, representing terms like "William," "Bernstein," and "Oregon." Word2Vec vectors are calculated for all 23 words individually.

Next, the vectors will be summed up and divided by the total number of words in the description (n, which in this case is 23). This operation results in a vector representation denoted as v1, computed as follows:

$$v_1 = \frac{\sum_{i=1}^{N} Word2Vec(w_i)}{N}$$

Formula 1. Average Word2Vec vector calculation.

Here, the vectors are in a D-dimensional space, where D equals 300.

Where N is the number of words in the reviews and v_1 is a vector representation of the reviews for place 1.

This method computes the average Word2Vec representation for the given reviews text. Similarly, other restaurants reviews can be converted into vectors using the same approach. This formula calculation was implemented in form of a Python function.

```python
def average_word2vec_vectors(x):

    # Creating a list for storing the vectors from the text of reviews
    global word_embeddings
    word_embeddings = []

    # Reading the each book description
    for line in grouped_reviews_df['cleaned']:
        avgword2vec = None
        count = 0
        for word in line.split():
            if word in google_model.wv.index_to_key:
                count += 1
                if avgword2vec is None:
                    avgword2vec = google_model.wv[word]
                else:
                    avgword2vec = avgword2vec + google_model.wv[word]

        if avgword2vec is not None:
            avgword2vec = avgword2vec / count

            word_embeddings.append(avgword2vec)
```

Figure 47. Python function for calculating average Word2Vec of places reviews.

A Python place recommendations function is implemented by choosing and printing 5 first vectors that are closest to the input place vector by cosine similarity.

```python
def recommendations(place_id):

    average_word2vec_vectors(grouped_reviews_df)

    # finding cosine similarity for the vectors
    cosine_similarities = cosine_similarity(word_embeddings)

    place = grouped_reviews_df[['place_ids']]
    #Reverse mapping of the index
    indices = pd.Series(grouped_reviews_df.index, index = grouped_reviews_df['place_ids']).drop_duplicates()

    idx = indices[place_id]
    sim_scores = list(enumerate(cosine_similarities[idx]))
    sim_scores = sorted(sim_scores, key = lambda x: x[1], reverse = True)
    sim_scores = sim_scores[1:6]
    places_indices = [i[0] for i in sim_scores]
    recommend = place.iloc[places_indices]
    for index, row in recommend.iterrows():
        print(row['place_ids'])
```

Figure 48. Python function for calculating similarities and returning a list of recommendations, 5 most similar places to the provided place.

5.6 TF-IDF Word2Vec

Term Frequency-Inverse Document Frequency(TF-IDF) is a numerical statistic used in information retrieval and text mining to measure the importance of a word in a document relative to a collection of documents (corpus). TF-IDF operates by assessing how frequently a word appears in a specific document in relation to its prevalence across the entire document collection. Essentially, it gauges the relevance of a word within a particular document. Words that are prominent in a single document or a small set of documents tend to have higher TF-IDF scores compared to common words like articles and prepositions, making it easier to identify significant terms in specific contexts [24].

Term Frequency (TF): This component measures how frequently a term (word) appears in a document. It is calculated by counting the number of occurrences of a word in the document. The idea is that words appearing more frequently in a document are more important.

$$TF(t,\ d)\ =\ \frac{Number\ of\ times\ term\ t\ appears\ in\ document\ d}{Total\ number\ of\ terms\ in\ document\ d}$$

Formula 2. Term Frequency calculation.

Inverse Document Frequency (IDF): IDF measures how important a term is across a collection of documents. Words that appear in many documents are usually less significant, and IDF penalizes these common words. IDF is calculated using the logarithm of the inverse fraction of documents containing the term.

$$IDF(t,\ D)\ =\ log(\frac{Total\ number\ of\ documents\ in\ the\ corpus\ |D|}{Number\ of\ documents\ containing\ term\ t+1})$$

Formula 3. Inverse Document Frequency calculation.

TF-IDF Score for a term in a specific document is calculated by multiplying TF and IDF:

$$TF\ -\ IDF(t,d,D)\ =\ TF(t,d)\ *\ IDF(t,D)$$

Formula 4. Term Frequency-Inverse Document Frequency calculation.

5.6.1 Combining TF-IDF Score with Word2Vec

Combining TF-IDF (Term Frequency-Inverse Document Frequency) scores with Word2Vec presents a powerful technique for recommendation systems.
TF-IDF, emphasizing word importance within a document and across a corpus, is multiplied by the Word2Vec vector representation of the word. This fusion enables a nuanced understanding of document content by considering both the frequency of words and their semantic context.

The steps for combining TF-IDF scores with Word2Vec vectors are the following:

1. Calculate TF-IDF Vectors: Compute the TF-IDF vectors for each word in the description. These vectors, denoted as $TF - IDF_1, TF - IDF_2, TF - IDF_N$, represent the TF-IDF scores for each word in the description. Note that TF-IDF vectors are not D-dimensional.

2. Calculate Word2Vec Vectors: Compute the Word2Vec vectors for each word in the description using the Word2Vec model.

3. Multiply TF-IDF and Word2Vec Vectors: Multiply the TF-IDF score and Word2Vec vector representation for each word. This results in a set of weighted Word2Vec vectors.

4. Total the Weighted Vectors: Sum up all the weighted Word2Vec vectors obtained in the previous step.

5. Divide by Sum of TF-IDF Vectors: Divide the total from step 4 by the sum of the TF-IDF vectors ($TF - IDF_1, TF - IDF_2, TF - IDF_N$).

$$v_1 = \frac{\sum_i^N TF{-}IDF_i * Word2Vec(w_i)}{\sum_i^N TF{-}IDF_i}$$

Formula 5. TF-IDF Word2Vec vector calculation.

The result is the vector representation of the given place reviews using TF-IDF Word2Vec.

The process combines the advantages of both TF-IDF and Word2Vec, providing a weighted representation of words based on their importance in the specific description.

```
#Building TFIDF model and calculate TFIDF score
tfidf = TfidfVectorizer(analyzer='word', ngram_range=(1, 3), min_df = 5, stop_words='english')
tfidf.fit(grouped_reviews_df['cleaned'])

# Getting the words from the TF-IDF model
tfidf_list = dict(zip(tfidf.get_feature_names_out(), list(tfidf.idf_)))
tfidf_feature = tfidf.get_feature_names_out() # tfidf words/col-names


# Building TF-IDF Word2Vec

# Storing the TFIDF Word2Vec embeddings
tfidf_vectors = []
line = 0

for reviews_text in corpus:
    sent_vec = np.zeros(300)

    weight_sum = 0

    for word in reviews_text:
        if word in google_model.wv.index_to_key and word in tfidf_feature:
            vec = google_model.wv[word]
            tf_idf = tfidf_list[word] * (reviews_text.count(word) / len(reviews_text))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_vectors.append(sent_vec)
    line += 1
```

Figure 49. Python function for calculating TF-IDF score combined with Word2Vec.

Similarly to the previous recommendations function the TF-IDF place recommendations function is implemented by choosing and printing 5 first vectors that are closest to the input place vector by cosine similarity.

```
def recommendations_tfidf(place_ids):

    # finding cosine similarity for the vectors

    cosine_similarities = cosine_similarity(tfidf_vectors,  tfidf_vectors)

    # taking the place_ids
    places = grouped_reviews_df[['place_ids', 'name', 'reviews_text']]
    #Reverse mapping of the index
    indices = pd.Series(grouped_reviews_df.index, index = grouped_reviews_df['place_ids']).drop_duplicates()

    idx = indices[place_ids]
    sim_scores = list(enumerate(cosine_similarities[idx]))
    sim_scores = sorted(sim_scores, key = lambda x: x[1], reverse = True)
    sim_scores = sim_scores[1:6]
    places_indices = [i[0] for i in sim_scores]
    recommend = places.iloc[places_indices]
    for index, row in recommend.iterrows():
        print(row['place_ids'], row['name'])
```

Figure 50. Python function for calculating similarities for TF-IDF vectors and returning a list of recommendations, 5 most similar places to the provided place.

5.7 Making the recommendations

When a user inputs a place they like, the system attempts to find similar places by using the provided place ID as input for the previously defined function. This function utilizes the place ID to retrieve textual reviews associated with the place. These reviews are then used to calculate the cosine similarity measure, enabling the system to identify places similar to the user's preference. The recommendations can be tested by providing one of the entries from the textual reviews dataset. The two algorithms were tested on one of the Mexican-style restaurants in the dataset.

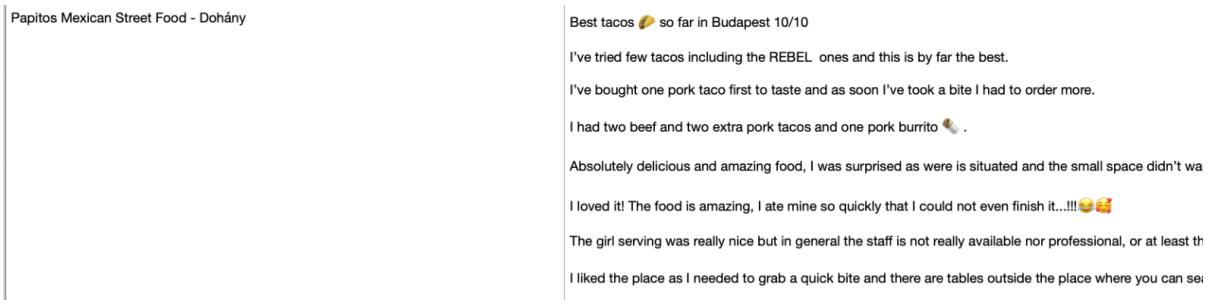


Figure 51. Textual reviews dataset restaurant entry example.

5.7.1 Average Word2Vec recommendations results

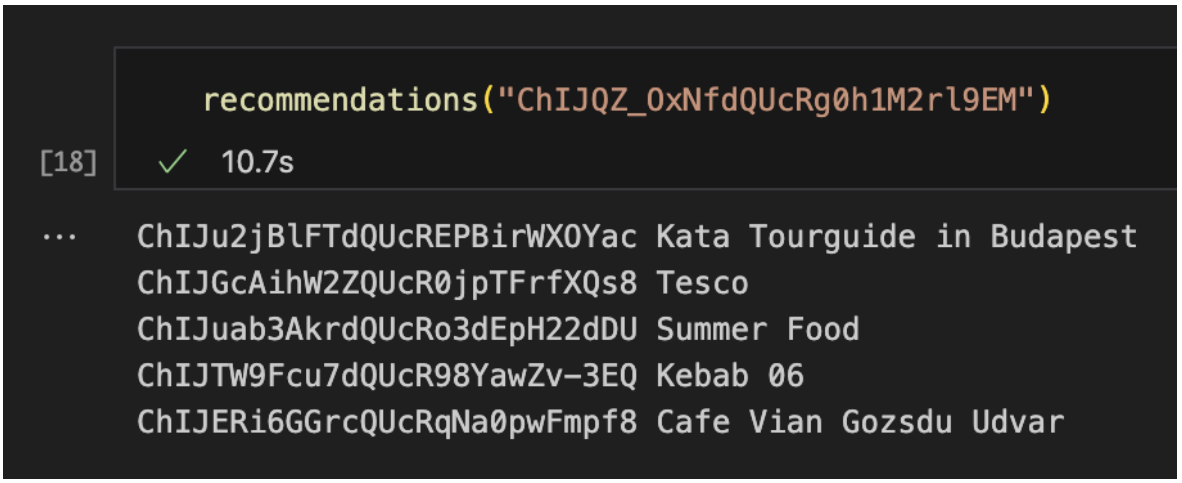


Figure 52. Average Word2Vec recommendations function call example.

5.7.2 TF-IDF Word2Vec recommendations results

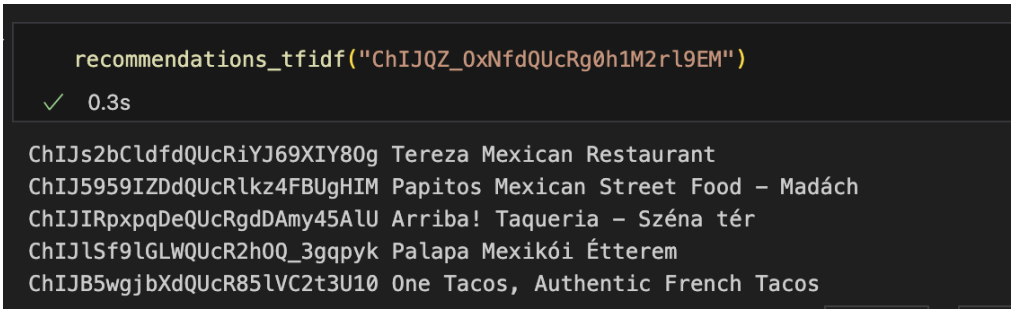


Figure 53. TF-IDF Word2Vec recommendations function call example.

5.7.3 Explaining results

In our specific scenario, employing a combination of TF-IDF and word2vec has proven to be a superior approach for providing relevant place recommendations compared to averaging word2vec embeddings alone. This choice is supported by the following reasoning:

1. Semantic Enrichment: TF-IDF (Term Frequency-Inverse Document Frequency) captures the importance of words in individual reviews, highlighting significant terms. Combining it with word2vec, which understands semantic relationships between words, enriches the context of the reviews. This semantic enrichment ensures a more nuanced understanding of the textual data, leading to more accurate recommendations.

2. Contextual Relevance: By incorporating both TF-IDF and word2vec, the hybrid approach not only considers the overall significance of words (TF-IDF) but also understands the contextual meaning of words (word2vec embeddings). This dual perspective enables the system to generate recommendations that align not only with individual word importance but also with the overall context of the reviews, resulting in more relevant suggestions for users.

3. Addressing Word Variability: Averaging word2vec embeddings might overlook the variability in word meanings, leading to generalized representations. Combining TF-IDF with word2vec captures the specific importance of words in different contexts, allowing the system to differentiate between words with multiple meanings. This differentiation leads to more precise and contextually accurate recommendations.

In summary, the integration of TF-IDF and word2vec provides a comprehensive understanding of the textual data, considering both word significance and semantic context. This comprehensive approach enhances the system's ability to discern relevant information, resulting in more precise and tailored recommendations for users.

6. Context-Aware Recommender Systems

Context-Aware Recommender Systems are designed to address a key limitation in traditional recommendation systems. While traditional systems focus on personalized suggestions based on user preferences, they often miss considering the user's context within the platform [25]. Context-aware recommendation systems fill this gap by taking into account various contextual factors such as time, location, device, weather, social context, and user activity. By incorporating these elements, these systems provide recommendations that are not only personalized but also highly relevant and timely, significantly enhancing the user experience.

In the context of recommending places, contextual filtering becomes especially valuable. It allows the system to display restaurants that are nearby or currently open, ensuring that the recommendations align with the user's immediate needs and location. This targeted approach enhances the user's overall satisfaction by offering practical and pertinent suggestions.

There are three main contextual filtering techniques: Contextual Pre-Filtering, Contextual Post-Filtering, and Contextual Modeling [25]. Among these, Contextual Modeling stands out as a more intricate approach, requiring advanced machine learning techniques like deep learning or reinforcement learning to decipher complex patterns within the data. This complexity extends to the need for higher customization and fine-tuning to ensure recommendations align closely with the user's specific dining context.

For our specific use case, opting for either Contextual Pre-Filtering or Post-Filtering proves advantageous. These methods provide effective ways to enhance our existing recommender system without delving into the complexities associated with contextual modeling. Pre-Filtering allows us to filter restaurant options based on specific criteria before generating recommendations, while Post-Filtering refines the recommendations after they are generated. Both approaches seamlessly integrate with our current system, making them practical and straightforward choices. Additionally, they offer the flexibility to adapt to varying user preferences and contextual factors, ensuring a user-friendly and tailored restaurant recommendation experience.

6.1 Contextual Pre-Filtering

Contextual Pre-Filtering in recommendation systems enhances the relevance of restaurant suggestions by applying specific filters based on contextual information before generating personalized recommendations. Unlike post-filtering, where recommendations are filtered after generation, contextual Pre-Filtering incorporates contextual data early in the recommendation process. This approach ensures that restaurant suggestions are accurately tailored to the user's current situation.

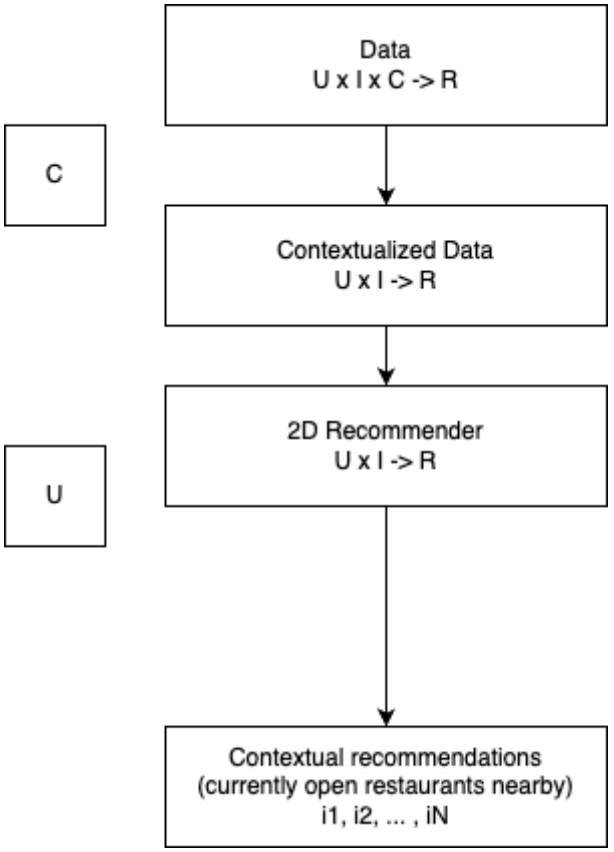


Figure 54. Contextual Pre-Filtering steps.

To implement contextual Pre-Filtering, a user-item-context tensor containing ratings is utilized. In traditional collaborative filtering systems, only a user-item interaction matrix with ratings is used. Contextual Pre-Filtering involves applying context-based filters before the traditional user-item interaction matrix recommendation system. For example, when recommending restaurants online, the algorithm might filter out options that don't match the desired context, such as restaurants that are not open during the user's preferred dining hours. To prevent over-specifying the context and addressing sparsity issues, context

generalization can be applied. Instead of specifying an exact time, a broader timeframe like "weekdays" can be considered [25].

After filtering the data by context, it reverts to the conventional user-item interaction matrix, used in traditional collaborative filtering recommendation systems. The contextualized data, filtered by specific contextual dimensions, represents a subset of the original data. This contextualized data undergoes a standard two-dimensional collaborative filtering recommendation system. The advantage is that there's no need to develop new algorithms to handle multidimensional contexts in the input tensor. The recommendation system simultaneously learns user and restaurant embeddings, which are then used to make predictions. The user vector is applied to the learned embeddings to predict ratings for restaurants the user hasn't interacted with yet. Finally, for contextual Pre-Filtering, the specified top number of restaurant recommendations are provided to the user.

6.2 Contextual Post-Filtering

Contextual Post-Filtering simplifies the process of enhancing recommendations by incorporating contextual information after the initial recommendations have been generated. Initially, there is a multidimensional dataset including user preferences, restaurant details, and various contextual factors.

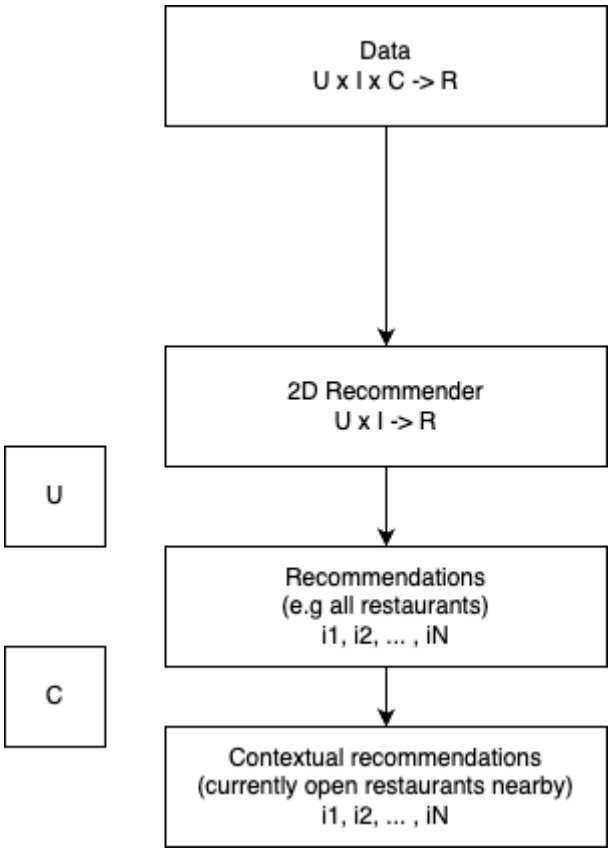


Figure 55. Contextual Post-Filtering steps.

In this approach, the original data is passed through a traditional recommendation system without considering the contextual aspects. This simplifies the process, leveraging standard collaborative filtering techniques [25]. The result is a set of recommendations, but these are the same as if there is no contextual data.

To tailor these generic recommendations to the specific context, the contextual information is applied afterward. This adjustment step, called Post-Filtering, involves refining the recommendations based on the current context. For instance, if a user typically prefers Italian cuisine on weekends, the Post-Filtering process can eliminate non-Italian restaurant options from the recommendations generated by the initial system. This ensures that the final recommendations align with the user's preferences within the given context, fulfilling the goal of providing personalized and relevant restaurant suggestions.

6.2.1 Applying contextual Post-Filtering to the TF-IDF Word2Vec recommendations

In this step, contextual post-filtering is implemented on the TF-IDF Word2Vec recommendations. This is achieved by adjusting the TF-IDF Word2Vec function and refining the recommendations to include only places located in the 10th district of Budapest.

```python
def recommendations_tfidf(place_ids):
    # Finding cosine similarity for the vectors
    cosine_similarities = cosine_similarity(tfidf_vectors, tfidf_vectors)

    # Extracting place information
    places = grouped_reviews_df[['place_ids', 'name', 'reviews_text', 'postal_code']]

    # Reverse mapping of the index
    indices = pd.Series(grouped_reviews_df.index, index=grouped_reviews_df['place_ids']).drop_duplicates()
    idx = indices[place_ids]

    # Calculating similarity scores
    sim_scores = list(enumerate(cosine_similarities[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:6]
    places_indices = [i[0] for i in sim_scores]

    # Postfiltering based on postal codes
    filtered_recommendations = []
    for index in places_indices:
        place_id = places.iloc[index]['place_ids']
        postal_code = places.iloc[index]['postal_code']
        if str(postal_code)[:2] == '10':
            filtered_recommendations.append(place_id)

    # Displaying the filtered recommendations
    for place_id in filtered_recommendations:
        place_info = places.loc[places['place_ids'] == place_id]
        print(place_info['place_ids'].values[0], place_info['name'].values[0])
```

Figure 56. Contextual Post-Filtering steps.

After applying the Contextual Post-Filtering the recommendation function returns places similar to the input place located in the specified district, in this case it is 10th.

```
recommendations_tfidf("ChIJQZ_0xNfdQUcRg0h1M2rl9EM")
✓ 0.0s
ChIJIRpxpqDeQUcRgdDAmy45AlU Arriba! Taqueria – Széna tér
```

Figure 57. The returned result after applying Contextual Post-Filtering.

7. Development Opporunities

In this chapter development opportunities for the explored recommender system are explored. The focus is on identifying and examining opportunities that can enhance the system's effectiveness and user experience.

Integration of Content-Based and Collaborative Filtering Techniques

In the landscape of recommender systems, a promising development opportunity arises from combining content-based recommendation methods and collaborative filtering into a unified hybrid recommendation system. This approach uses the strengths of both systems, utilizing content analysis and user similarity patterns to deliver recommendations that are more precise and tailored to individual preferences, as well as featuring more diversity in recommendations [4].

Application of Reinforcement Learning for Continuous Improvement

Another opportunity for development lies in the integration of reinforcement learning. This presents an opportunity to establish a feedback loop within the recommender system. By leveraging reinforcement learning, the system can continuously learn and adapt based on users' interactions, leading to a constant improvement in recommendations. This adaptive mechanism ensures that the recommendations remain relevant and evolve over time, aligning more closely with the changing preferences of individual users [7].

8. Summary

The research paper delves into the exploration of general recommendation techniques and approaches, offering insights into the comprehensive process of data preprocessing and the key steps involved in building a recommender system. The study provides a practical example, illustrating these processes using the RandomForest algorithm as the classifier. By focusing on this algorithm, the paper aims to provide a clear and applicable demonstration of the fundamental steps in constructing an effective recommender system.

The integration of textual data analysis and contextual filtering techniques in recommendation systems was explored. Word2Vec, a natural language processing tool, was used for textual review analysis.

Two methods were utilized: average Word2Vec and TF-IDF Word2Vec. The former generated establishment review vectors by averaging Word2Vec word vectors but lacked detail. The latter combined TF-IDF scores with Word2Vec vectors, offering precise and contextually relevant recommendations.

Results indicated that TF-IDF Word2Vec outperformed average Word2Vec, delivering accurate personalized recommendations considering word importance and semantic context.

Contextual Pre-Filtering and Contextual Post-Filtering techniques were applied. Pre-Filtering refined initial options based on user preferences, while Post-Filtering adjusted recommendations according to specific contexts, enhancing relevance.

The study highlighted that integrating TF-IDF scores with Word2Vec vectors and applying contextual filtering techniques significantly improved recommendation system effectiveness, delivering tailored and contextually relevant suggestions for users.

The development opportunities include combining the techniques used and collaborative filtering into a hybrid recommendation system. This system combines the explored content-based recommender system approach and collaborative filtering's search for user

similarities patterns. Additionally, by applying reinforcement learning, there is an opportunity to establish a feedback loop for constant improvement of the recommendations based on individual users' interactions with the recommender system.

9. References

[1] Pathak B, Garfinkel R, Gopal R, Venkatesan R, Yin F. Empirical analysis of the impact of recommender systems on sales, 2010.
Retrieved November 4, 2023 from
https://www.researchgate.net/publication/220591872_Empirical_Analysis_of_the_Impact_of_Recommender_Systems_on_Sales

[2] Pu P, Chen L, Hu R. A user-centric evaluation framework for recommender systems, 2011.
Retrieved November 4, 2023 from
https://www.researchgate.net/publication/228372156_A_User-Centric_Evaluation_Framework_of_Recommender_Systems

[3] Burke R. Knowledge-Based Recommender Systems, 2000.
Retrieved November 5, 2023 from
https://www.researchgate.net/publication/2378325_Knowledge-Based_Recommender_Systems

[4] Burke R. Hybrid recommender systems: survey and experiments, 2002.
Retrieved November 5, 2023 from
https://www.researchgate.net/publication/263377228_Hybrid_Recommender_Systems_Survey_and_Experiments

[5] F.O. Isinkaye, Y.O. Folajimi, B.A. Ojokoh. Recommendation systems: Principles, methods and evaluation, 2015.
Retrieved November 5, 2023 from
https://www.researchgate.net/publication/283180981_Recommendation_systems_Principles_methods_and_evaluation

[6] Shyong K, Frankowski D, Riedl J. Do you trust your recommendations? An exploration of security and privacy issues in recommender systems, 2006.
Retrieved November 7, 2023 from
https://files.grouplens.org/papers/lam-etrics2006-security.pdf

[7] Xiaocong Chen, Lina Yao, Julian McAuley, Guanglin Zhou, Xianzhi Wang. Deep reinforcement learning in recommender systems: A survey and new perspectives, 2023.
Retrieved December 11, 2023 from
https://www.sciencedirect.com/science/article/pii/S0950705123000850

[8] Rich Caruana, Alexandru Niculescu-Mizil. An Empirical Comparison of Supervised Learning Algorithms, 2006.
Retrieved November 12, 2023 from
https://www.cs.cornell.edu/~caruana/ctp/ct.papers/caruana.icml06.pdf
[9] J. R. Quinlan. Induction of Decision Trees, 1986.
Retrieved November 18, 2023 from
https://link.springer.com/article/10.1007/BF00116251
[10] Lior Rokach, Oded Maimon. Top-down induction of decision trees classifiers-a survey, 2005.
Retrieved November 18, 2023 from
https://www.researchgate.net/publication/3421651_Top-Down_Induction_of_Decision_Trees_Classifiers-A_Survey
[11] Shalev-Shwartz Shai, Ben-David Shai. Understanding Machine Learning: From Theory to Algorithms, 2014.
Retrieved November 18, 2023 from
https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning/understanding-machine-learning-theory-algorithms.pdf
[12] Leo Breiman. Bias, variance, and arcing classifiers, 1996.
Retrieved November 12, 2023 from
https://www.stat.berkeley.edu/~breiman/arcall96.pdf
[13] Zhou Zhi-Hua. Ensemble Methods: Foundations and Algorithms, 2012.
Retrieved November 12, 2023 from
http://didattica.cs.unicam.it/old/lib/exe/fetch.php?media=didattica:magistrale:ml:ay_1920:ensemble_methods_zhou.pdf
[14] Breiman L. Bagging Predictors, 1996.
Retrieved November 18, 2023 from
http://didattica.cs.unicam.it/old/lib/exe/fetch.php?media=didattica:magistrale:ml:ay_1920:ensemble_methods_zhou.pdf
[15] Ho, Tin Kam. Random Decision Forests, 1995.
Retrieved November 18, 2023 from
http://vision.cse.psu.edu/seminars/talks/2009/random_tff/odt.pdf

[16] Ho, Tin Kam. The Random Subspace Method for Constructing Decision Forests, 1998.
Retrieved November 18, 2023 from
https://ieeexplore.ieee.org/document/709601
[17]Hastie, Trevor, Tibshirani, Robert, Friedman, Jerome. The Elements of Statistical Learning, 2008.
Retrieved November 18, 2023 from
https://www.sas.upenn.edu/~fdiebold/NoHesitations/BookAdvanced.pdf
[18] Piryonesi S. Madeh, El-Diraby Tamer E. Role of Data Analytics in Infrastructure Asset Management: Overcoming Data Size and Quality Problems, 2020.
Retrieved November 18, 2023 from
https://www.researchgate.net/publication/341797131_Role_of_Data_Analytics_in_Infrastructure_Asset_Management_Overcoming_Data_Size_and_Quality_Problems
[19] Piryonesi S. Madeh, El-Diraby Tamer E. Climate change impact on infrastructure: A machine learning solution for predicting pavement condition index, 2021.
Retrieved November 18, 2023 from
https://www.sciencedirect.com/science/article/pii/S0950061821026568
[20] Hardesty Larry. Explained: Neural networks, 2017.
Retrieved November 18, 2023 from
https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414
[21] Snezana Agatonovic-Kustrin, Rod Beresford. Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research, 2000.
Retrieved November December 11, 2023 from
https://www.sciencedirect.com/science/article/pii/S0731708599002721
[22] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space, 2013.
Retrieved October 29, 2023 from
https://www.researchgate.net/publication/234131319_Efficient_Estimation_of_Word_Representations_in_Vector_Space
[23] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality, 2013.
Retrieved October 29, 2023 from
https://www.researchgate.net/publication/257882504_Distributed_Representations_of_Words_and_Phrases_and_their_Compositionality

[24] Juan Ramos. Using TF-IDF to Determine Word Relevance in Document Queries, 2003.
Retrieved November 9, 2023 from
https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=b3bf6373ff41a115197cb5b30e57830c16130c2c
[25] Gediminas Adomavicius, Bamshad Mobasher, DePaul University, Francesco Ricci, Alexander Tuzhilin. Context-Aware Recommender Systems, 2011.
Retrieved November 9, 2023 from
https://www.researchgate.net/publication/220605653_Context-Aware_Recommender_Systems

10. Figure Index