FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	AutoML approach in recommendation systems
Student:	Daniil Pastukhov
Supervisor:	Ing. Stanislav Kuznetsov
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

1. Describe the problem of AutoML in the recommendation systems.
2. Choose appropriate state-of-the-art algorithms with a different approach like k-NN, Matrix Factorization or Autoencoders.
3. Focus on recall, catalogue-coverage, serendipity and diversity metrics.
4. Provide data preprocessing at Movie database.
5. Prepare models and experiments with several AutoML algorithms.
6. Evaluate models and discuss the results of the experiments.

Electronically approved by Ing. Karel Klouda, Ph.D. on 17 February 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

AutoML approach in recommendation systems

Daniil Pastukhov

Department of Applied Mathematics
Supervisor: Ing. Stanislav Kuznetsov

May 13, 2021

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 13, 2021 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Pastukhov, Daniil. *AutoML approach in recommendation systems.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Ve všech oblastech strojového učení, stejně jako v doporučovacích systémech, není snadné rozhodnout, který model by měl být v daném kontextu použit. Doporučovací systémy jsou softwarové nástroje, které se snaží předpovědět, o jaké položky by měl uživatel zájem. Cílem této práce je vyhodnotit různé přístupy AutoML s využitím nejmodernějších algoritmů a metrik, jako recall, pokrytí katalogu a serendipity. AutoML je proces automatizace časově náročného a iterativního procesu trénování ML modelu. V AutoML existují dva vhodné přístupy, které jsme si vybrali k experimentování - hyperparametrická optimalizace a metaučení. V této práci jsme testovali nejmodernější algoritmy na veřejně dostupných datových sadách MovieLens s využitím technik AutoML. Také jsme navrhli alternativní definici serendipity.

Klíčová slova doporučovací systémy, serendipity, strojové učení, automatizované strojové učení, automl

Abstract

In all machine learning domains, as well as in recommendation systems, it is not easy to decide which model should be used in a given context. Recommendation systems are software tools trying to predict what items a user would be interested in. This thesis aims to evaluate different AutoML approaches using state-of-the-art algorithms and metrics, such as recall, catalogue coverage, and serendipity. AutoML is a process of automating the time-consuming and iterative training process of the ML model. There two relevant approaches in AutoML that we have chosen to experiment with — Hyperparameter optimization and Meta-Learning. In this thesis, we tested state-of-the-art algorithms on publicly available MovieLens datasets employing AutoML techniques, and also proposed the alternative definition of serendipity.

Keywords recommendation systems, serendipity, machine learning, automated machine learning, automl

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we will briefly describe our motivation and problem statement and determine the goals for this work.

1.1 Motivation

Recommendation systems (RS) are software tools trying to predict what items a user would be interested in. They are widely used in various domains, e.g., music and video services (Spotify[1], YouTube[2]), e-commerce (Amazon[3]), social media (Facebook[4]). RS help stores to promote their products, allowing users to easier discover them. It is vital to suggest relevant items since irrelevant recommendations may be frustrating and may deteriorate user satisfaction. However, relevance is not the only major concept, as a user may have already seen relevant suggestions, and, perhaps, he expects to see something novel and unusual. To achieve more diverse and unexpected recommendations, we can focus on serendipity rather than on accuracy. Generally, *serendipity* means "unexpected discovery" or "fortunate chance"; hence, serendipitous recommendations can intuitively lead to diversification and catalogue coverage improvement.

Moreover, the experts spend much time on data processing, choosing a proper algorithm, and optimizing it. Automated machine learning (AutoML) may come in helpful and can be used to address these problems, allowing developers to spend their time more productive and efficient. Furthermore, since different domains may benefit from specific algorithms, choosing a suitable algorithm requires additional knowledge about the field, which is not always present. AutoML reduces the required level of knowledge for obtaining an accurate model.

[1] `https://www.spotify.com`
[2] `https://www.youtube.com`
[3] `https://www.amazon.com`
[4] `https://www.facebook.com`

1.2 Problem statement

The first problem is the serendipity evaluation of RS. There is no straightforward and conventional way to measure the serendipity of recommendations provided by RS; thus, the comparison, in terms of serendipity, of different recommendation approaches is difficult. In this work, we will propose a new definition of serendipity based on previous studies.

The second problem is a fine-tuning of a recommendation algorithm. Fine-tuning allows us to fit the model for the concrete area so that it can perform better. Hyperparameters may be different for distinct domains; therefore, the automation of this process is desirable. In terms of this work, we will try to facilitate a fine-tuning of recommendation algorithms.

The third problem is a time cost for fine-tuning an algorithm. Ideally, we would like to use previous experience to accelerate the optimization process. In this thesis, we propose a meta-learning approach for such a purpose.

1.3 Goals of this thesis

We set six goals for this thesis:

1. Describe the problem of AutoML in the recommendation systems.
2. Choose appropriate state-of-the-art algorithms with a different approach like k-NN, Matrix Factorization or Autoencoders.
3. Evaluate approaches using different metrics, like recall, catalogue coverage, serendipity, or diversity.
4. Provide data preprocessing at Movie database.
5. Prepare models and experiments with several AutoML algorithms.
6. Evaluate models and discuss the results of the experiments.

Chapter 2

Background

In this chapter, we will give an introduction to the area of Recommendation Systems (RS) and Machine Learning (ML). We will describe different RS approaches and introduce Automated Machine Learning (AutoML). Then we will focus on RS evaluation and different metrics. Finally, we will define multi-objective optimization and describe the related work.

2.1 Theoretical background

2.1.1 RS approaches

The approaches in RS are commonly divided into three groups: *collaborative filtering*, *content-based filtering* and *hybrid recommender systems* [1].

2.1.1.1 Collaborative filtering

Collaborative filtering (CF) is a technique that considers user interests and provides recommendations by comparing them with other users' interests. This method usually results in better performance compared to the different approaches and is frequently used since it does not require any information about the items themselves; the only thing it needs is a matrix containing user profiles. Let us show an example of how collaborative filtering works. A user likes *Star Wars* and *Avengers* movies, and he is looking for a new film to watch. His neighbourhood, i.e., similar users, liked *The Dark Knight* and *Blade Runner* movies, so we can consider recommending them. There are two types of CF techniques: *model-based filtering* and *memory-based filtering*.

Model-based filtering is an approach where the previous experience is used to develop a model to improve the CF technique's performance. Different data mining and machine learning techniques may be employed in a model

building process, for example, Neural Networks, Bayesian networks, clustering methods, and association rules.

Memory-based filtering is an approach, which uses user rating data to compute the similarity between users or items. There are two memory-based methods: *user-based* and *item-based*.

2.1.1.2 Content-based filtering

Content-based filtering (CBF) is a technique that uses item features to recommend other items similar to what the user likes, based on his previous actions or explicit feedback. This method tends to emphasize items that are similar to the ones a user has positively rated. CBF uses different approaches to measure similarity and make relevant recommendations, such as TF-IDF, probabilistic models. In contrast to collaborative filtering, this method analyzes the content of each item. As a result, thorough knowledge of the items is required. Let us show an example considering the same user described in section 2.1.1.1, that likes *Star Wars* and *Avengers*. *Star Wars* is a popular science fiction movie, and *Avengers* is a popular action superhero movie. So content-based recommendation engine can propose *Avatar* and *Iron Man* since these two movies are similar to ones the user likes.

2.1.1.3 Hybrid recommender systems

Hybrid recommender systems are a combination of the approaches discussed before. Such systems combine the advantages of both CF and CBF methods. It takes into account information about both users and items; thus, such systems are "smarter" than CF and CBF ones.

2.1.2 Machine learning

Machine learning (ML) is an application of artificial intelligence that gives computers the ability to learn and act as humans do, and improve their performance over time, by providing them with more data and information, which can be presented as observations or real-world interactions. In other words, machine learning is the science of getting computers to act without being programmed to do so. In machine learning, algorithms are trained to find patterns and features in large amounts of data to make decisions and predictions based on new data. The better the algorithm, the more accurate the decisions and predictions will become as it processes more data [3]. At present, machine learning helps to solve problems in various areas and often outperforms humans. In medicine, doctors use systems that provide preliminary results, e.g., a segmented X-Ray image, that may be further used in diagnosis determination. In banks, ML systems determine whether a person is loan eligible or

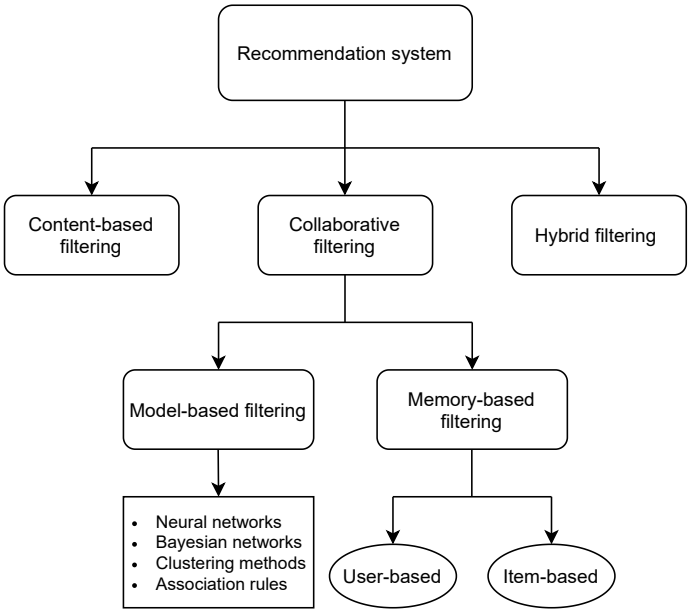

Figure 2.1: Types of recommendation systems. [2]

not. In email services, unwanted spam e-mails can be automatically filtered out using classification methods.

Machine learning pipeline usually consists of three stages: *Data Preparation*, *Model building and training*, and *Model deployment and monitoring*. Now, we will briefly describe details about them.

Data Preparation stage includes data gathering and data processing operations, such as normalization, transformation, validation, and featurization. The final step is data splitting, where the cleaned dataset is split into training and testing parts.

Model building and training stage includes algorithm selection and optimization. The type of algorithm depends on the problem (e.g., regression, classification) and the amount of available data. The training is an iterative process, where the algorithm parameters are adjusted by the optimization algorithm. The goal of training is to minimize the relative validation error and yield accurate predictions. The resulting trained algorithm is called a model. Different techniques like cross-validation and regularization are used to ensure the quality of a model and avoid overfitting.

Model deployment and monitoring is the final stage of the ML pipeline. The model is deployed in a production environment; its life cycle does not end. The world and the environment are constantly changing; thus, we need

to monitor changes and occasionally modify the existing model to keep the model up to date.

Every stage of an ML pipeline is vital for an ML project. Machine learning specialists commonly spend much time going through the above stages and have to do everything manually. To ease that process, we can try to automate it using automated machine learning, which will be discussed in the following subsection.

2.1.3 Automated machine learning

Automated machine learning (AutoML) has progressed dramatically over the past decade. More and more researchers focus on AutoML. Generally, AutoML is a process of automating the time-consuming and iterative training process of the ML model. AutoML allows ML and non-ML experts to build ML models with high efficiency and productivity, along with sustaining model quality. Utilizing AutoML allows a more straightforward development process. A few lines of code can generate the code necessary to begin developing a machine learning model. At the moment, many companies provide AutoML solutions, such as Google Cloud AutoML[5] or H2O Driverless AI[6]. They are capable of feature engineering, model fine-tuning and validation, and model deployment. For businesses, such systems may be really relevant. They do not need to hire ML experts, which can potentially save money, and still achieve reasonable results by using the existing AutoML system. The usual AutoML pipeline consists of data preparation, feature engineering, model generation, and model estimation steps [4].

The approaches in AutoML can be divided into three separate groups [5]: *Hyperparameter Optimization*, *Meta-Learning* (e.g., [6]), *Neural Architecture Search* (NAS) (e.g., [7]).

2.1.4 Hyperparameter optimization

A model hyperparameters are parameters that are used to control the learning process. Hyperparameters are usually user-specified, and their values cannot be estimated from data. We cannot know the best hyperparameters for a model for a given problem. Thus, we usually use heuristics or empirical rules to initialize them or find them using the trial and error method.

Now, we will briefly describe three methods that are used for hyperparameter optimization. Additionally, we need to keep in mind that the efficiency of the optimization method can be represented by the number of iterations required to find the optimal values. The less it is, the more efficient it is.

[5] `https://cloud.google.com/`

[6] `https://www.h2o.ai/products/h2o-driverless-ai/`

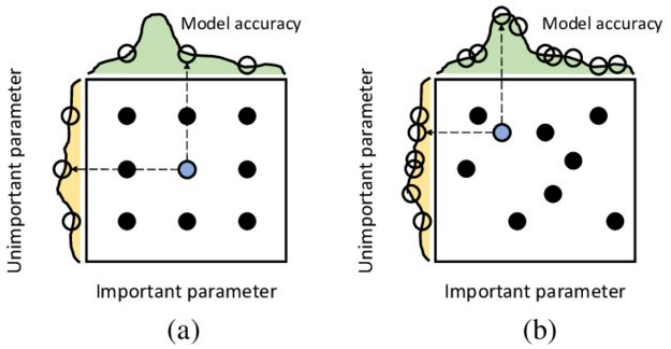

Figure 2.2: Comparison between (a) grid search; and (b) random search for hyperparameter tuning. The nine points denote the candidates. The curves on the left and on the top denote model accuracy (e.g. Normalized Mean Square Error) as a function of each search dimension. [8]

Grid search is an exhaustive searching, which tests all hyperparameter combinations on a user-predefined subset. It is a primitive and inefficient method.

Random search replaces an exhaustive testing of all combinations by selecting them randomly. It runs a fixed number of iterations or until the desired condition is not met. The difference between grid search and random search is shown in Figure 2.2.

Bayesian hyperparameter optimization is a method that is based on the statistical analysis of the objective function. Unlike random search, this method is an informed search. It keeps track of past evaluations, but that is also why we cannot run multiple processes simultaneously. The first step is building the probabilistic model, also known as the surrogate model, for the objective function $P(metric|hyperparameters)$. Such function gives the probability of maximizing/minimizing metric given the hyperparameters. There are different approaches to build a surrogate model, such as Gaussian process, Tree-structured Parzen Estimator [9]. The surrogate model is more straightforward to optimize than the actual objective function. Then we choose the hyperparameters that perform best on a surrogate model and test it on the objective function.

Evolutionary algorithm (EA) is a population-based optimization algorithm inspired by biological evolution. Crossover, selection, and mutation are basic mechanisms used in EA. The fitness function represents the quality of the individual and is used to compare them. The algorithm starts with the initialization of a population, which can be done randomly or heuristically.

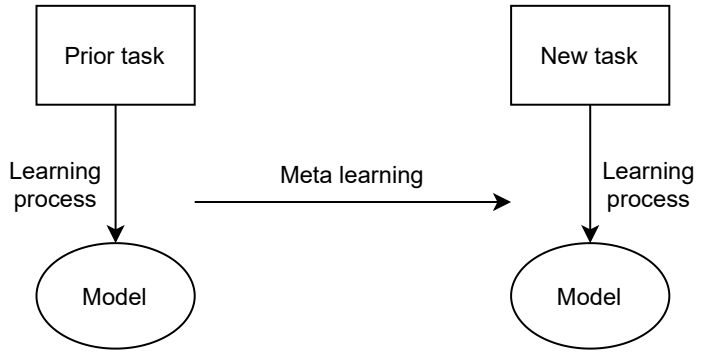

Figure 2.3: Meta learning concept.

It is followed by the primary cycle of calculating fitness function, performing mutations, crossover, and selection.

2.1.5 Meta-learning

"The term meta-learning covers any type of learning based on prior experience with other tasks." [5]

Meta-learning, or **learning to learn**, refers to a science of observing how different ML models perform on various tasks and then learning from this experience, which is also known *meta-data*, to speed up the learning process when facing a new task. We can use knowledge about already trained models used in the prior tasks in recommendation systems and apply them to the new tasks. For example, we have trained a model for a video hosting service, which performs well. We have extracted meta-data from the learning process: dataset properties and model hyperparameters. The new task is an RS for the music streaming website. Then meta-learning can be used in the initialization of hyperparameter optimization algorithm.

2.1.6 Evaluation of RS

Initially, most recommendation systems were evaluated by their ability to predict users' preferences accurately. However, accurate recommendations are now considered crucial but insufficient to deploy a good and valuable RS. Users often expect to see recommendations that do not precisely match their tastes. They may be curious and desire to see novel and exciting things. Thus, it is essential to study the domain and choose appropriate metrics to improve business indicators [10].

There are three different types of experiments: *offline* and *online* experiments; and *user studies*.

Offline experiments are performed on the historical precollected dataset that contains user ratings or interactions. Using such a dataset, we try to model the user's behavior that interacts with a recommendation system. This approach is based on the assumption that the user behavior at the data collection moment will be similar enough to the one after RS deployment.

User studies are conducted by a recruited set of test subjects, which are asked to perform specific tasks requiring interaction with the recommendation system. While the subjects perform the tasks, we observe and record their behavior, collecting quantitative measurements, such as what part of the task was completed, the accuracy of the task results, or the time taken to perform the task. Offline testing does not guarantee a reliable simulation of users' behavior, so the proper evaluation of RS requires real user interactions.

Online experiments are performed by the actual users that perform real tasks. Online evaluation provides the most substantial evidence as to the actual value of an RS. As opposed to offline experiments and user studies, this type of experiment is harder to conduct. It requires a functioning RS running in the production environment. This fact frequently makes online testing unaffordable.

2.1.7 Metrics

Accurate metrics are necessary for developing a thriving RS. There are different metrics that are significant for RS evaluation. Recall and precision belong to the information retrieval measure and helps us to determine how relevant recommendations are. Catalogue coverage, serendipity, and diversity are related concepts, which lead to a better user experience in RS. Additionally, CF and hybrid approaches require the computation of user similarities, which can be done, e.g., using Pearson correlation or cosine similarity.

2.1.7.1 Similarity measurement

In order to compare different users, we need to define a *similarity measure*. Let x and y be real-valued vectors, such that $x, y \in \mathbb{R}^n$, then a function $sim(x, y)$ that quantifies the similarity between x and y is called a *similarity function*.

Pearson correlation similarity of two users x, y is defined as follows:

$$sim(x,y) = \frac{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r_x})(r_{y,i} - \bar{r_y})}{\sqrt{\sum_{i \in I_{xy}} (r_{x,i} - \bar{r_x})^2} \sqrt{\sum_{i \in I_{xy}} (r_{y,i} - \bar{r_y})^2}} \tag{2.1}$$

where I_{xy} is a set of items that were rated by both x and y, $r_{x,i}$ is a rating of item i given by user x and $\bar{r_x}$ is an average rating given by user x to all items.

Cosine similarity between users x and y is defined as:

$$sim(x,y) = cos(\vec{x},\vec{y}) = \frac{\vec{x}\cdot\vec{y}}{\|\vec{x}\| \times \|\vec{y}\|} = \frac{\sum_{i\in I_{xy}} r_{x,i} r_{y,i}}{\sqrt{\sum_{i\in I_x} r_{x,i}^2}\sqrt{\sum_{i\in I_y} r_{y,i}^2}} \tag{2.2}$$

2.1.7.2 Precision, recall, and catalogue coverage

Precision is a fraction of relevant items that were recommended to a user. Precision for N recommendations can be formalized as:

$$precision@N = \frac{|R_u \cap I_u|}{|R_u|} \tag{2.3}$$

where I_u denotes all user interactions and R_u denotes the recommendation made by RS.

Recall is a fraction of relevant items that were recommended by RS. Formally, we can define recall for N recommended items as follows:

$$recall@N = \frac{|R_u \cap I_u|}{|I_u|} \tag{2.4}$$

Catalogue coverage represents how many items are possible to be recommended by the RS. Catalogue coverage is a valuable measure for massive catalogues, where many items are left unseen. We can define catalogue coverage the same way as a recall for N recommended items:

$$coverage@N = \frac{|\bigcup_{u\in U} R_u|}{|C|} \tag{2.5}$$

where the numerator denotes all items that were recommended to all users and C denotes all items in the catalogue.

2.1.7.3 Serendipity and diversity

Diversity is a concept, which describes how items are different with respect to each other. Diversity is relevant for systems with a great variety of items. Diverse recommendations may lead to a better user experience.

Preliminary to the definition of serendipity, we need to define three auxiliary concepts: *relevance*, *novelty* and *unexpectedness*.

Relevance is an essential property of the recommendations since irrelevant suggestions may lead to low user satisfaction and low recall. Hence, the overall performance of the recommendation system can become insufficient.

The item is considered relevant when the user may be interested in it. Let us assume that the user visits the e-shop and looks at the table. Then the relevant recommendation maybe a chair. In addition, we can keep in mind that the user's "neighbours" can also be considered in the decision process.

We can measure relevance by (2.6) finding the minimal distance between the new item and user interactions or (2.7) finding the mean distance between the new item and user interactions. We can also ignore old user interactions due to their possible obsoleteness.

$$rel(i,u) = \min_{j \in S \subseteq I_u} sim(i,j) \tag{2.6}$$

$$rel(i,u) = \frac{1}{|I_u|} \sum_{j \in S \subseteq I_u} sim(i,j) \tag{2.7}$$

where i is an item, u is a user profile, I_u is user interactions, S is either a subset of I_u or I_u itself, $sim(i,j)$ is any similarity metric (e.g., cosine similarity), normalized to [0, 1].

Novelty is a desirable feature in recommendation systems. Novel recommendations improve the catalogue coverage and let users discover more new things than they would find themselves.

The item is considered novel when the user has not either seen it or is familiar with it.

Vargas [11] proposed 2 ways of novelty metric: *popularity-based* (2.8) and *distance-based* (2.9).

$$nov(i) = 1 - p(i) \tag{2.8}$$

where $p(i)$ denotes the probability that the item i was rated by any user.

$$nov(i,u) = \min_{j \in S \subseteq I_u} dist(i,j) \tag{2.9}$$

where i is an item, u is a user profile, I_u is user interactions, S is either a subset of I_u or I_u itself and $dist(i,j)$ can be defined as follows:

$$dist(i,j) = 1 - sim(i,j)$$

Unexpectedness is one of the most essential concepts, which underlies serendipity, which grants a user an opportunity to discover more surprising items and expand the recommended categories. The measurement of unexpectedness is quite a challenging task, due to its varying definition. According to [12], there are four different definitions of unexpectedness.

In [13], the authors proposed measuring unexpectedness by comparing the recommendations of RS with the recommendations made by primitive prediction models (2.10). The main idea relies on predicting that the expected item is a trivial task, whereas predicting something unexpected is more demanding. Using such a method, we can see how our recommendations differ from the trivial ones.

$$unexp(i,u) = \frac{PR_u}{RS_u} \tag{2.10}$$

where PR_u is recommendations generated by the primitive model and RS_u is the recommendations generated by the recommendation system.

Serendipity is challenging to define a feature of recommendation systems and can be interpreted in 8 different ways [12]. The most common definition considers three properties - relevance, novelty, and unexpectedness - while it is still possible to ignore either relevance or novelty. Improvement of serendipity can be crucial for improving catalogue coverage and diversification [14].

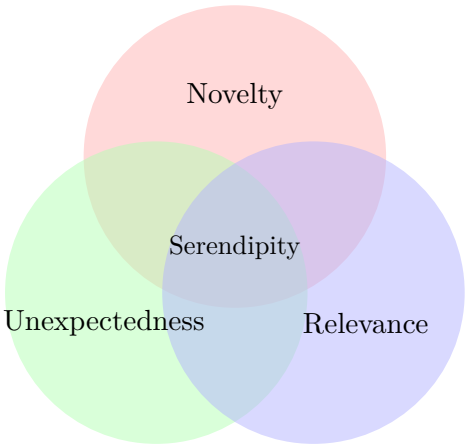

Figure 2.4: A Venn diagram describing serendipity concept.

Combining the three concepts described before, we can define the serendipity metric as follows:

$$serendipity(i,u) = \alpha * nov(i,u) + \beta * unexp(i,u) + \gamma * rel(i,u) \tag{2.11}$$

where $nov(i,u)$, $unexp(i,u)$, $rel(i,u)$ are metrics we have defines above and α, β, γ are coefficients that represent the importance of $nov(i,u)$, $unexp(i,u)$, $rel(i,u)$ metrics respectively. The equation $\alpha + \beta + \gamma \ = 1$ must be satisfied.

Both diversity and serendipity metrics are introduced to overcome the overfitting problem, i.e. when RS loses the ability to help users to discover new things.

2.1.8 Multi-objective optimization

In general, multi-objective optimization has several objective functions with subject to inequality and equality constraints to optimize [15]. The goal is to find a set of solutions that do not have any constraint violation (known as feasible solutions) and are as good as possible regarding all its objectives values. The problem definition in its general form is given by:

$$\begin{aligned} \min \quad & f_m(x) && m = 1, .., M \\ \text{s.t.} \quad & g_j(x) \leq 0 && j = 1, .., J \\ & h_k(x) = 0 && k = 1, .., K \\ & x_i^L \leq x_i \leq x_i^U && i = 1, .., N \end{aligned} \tag{2.12}$$

The formulation above defines a multi-objective optimization problem with N variables, M objectives, J inequality, and K equality constraints. Moreover, for each variable x_i, lower and upper variable boundaries (x_i^L and x_i^U) are defined. [16]

NSGA2 is a computationally efficient multi-objective evolutionary algorithm proposed by [17]. It uses an elitism principle, i.e., the elites of a population are given the opportunity to be carried to the next generation. Furthermore, it uses an explicit diversity preserving mechanism (crowding distance). The pseudocode can be found in Algorithm 1.

Algorithm 1: NSGA2 [18]

```
initialize a population of size N;
while Termination criteria is not met do
    Perform elitism selection technique;
    Perform genetic operations;
    Evaluate objectives;
    Perform fast non-dominated sorting;
    Calculate and assign crowding distance;
end
return best individuals;
```

2.2 Related work

In [14], authors studied a trade-off between serendipity and catalogue coverage. They concluded that an increase in serendipity leads to higher catalogue coverage. However, at the same time, an increase increase in catalogue coverage does not guarantee an increase in serendipity. They also discovered that

increasing serendipity may negatively impact recall, but using proper techniques, e.g., arrangement of recommendations, it is possible to avoid negative consequences.

Wang [19] proposed an AutoML architecture that is capable of handling data and finding the optimal recommendation model. The architecture consists of 4 blocks: Mapper, Interactor, Optimizer and Tuner. The first block, Mapper, is responsible for conversion the data into low-dimensional embeddings. Interactor is used to simulate different ways of interactions between entities. Optimizer manages the computation of metric and loss functions. And the last part of the architecture, Tuner, is used to find the optimal hyperparameters.

Anand [20] came up with an extension of existing recommender system library "Surprise". Their algorithm begins with evaluating baseline score with a random algorithm, and then each algorithm is optimized in parallel. Algorithms, that perform worse than the baseline after some number of iterations, are not optimized anymore. Hyperparameter optimization is handled by Hyperopt library [21], where Tree of Parzens Estimator (TPE), Adaptive TPE (ATPE) and Random Search optimization methods may be employed.

Chapter 3

Methodology

In this chapter, we will discuss the data and its preparation, algorithms and AutoML approaches we use in our experiments.

3.1 Data Understanding

The MovieLens datasets are the result of users interacting with the MovieLens online recommender system over the course of years. [22]

We have conducted our experiments on publicly accessible *MovieLens* datasets. The MovieLens datasets, first released in 1998, describe people's expressed preferences for movies. These preferences take the form of tuples, each resulting from a person expressing a preference (0–5 star rating) for a movie at a particular time. These preferences were entered by way of the MovieLens website - a recommender system that asks its users to give movie ratings to receive personalized movie recommendations. [22]

For our experiments, we have chosen MovieLens-100k and MovieLens-1m datasets. The specific details are shown in Table 3.1.

	MovieLens-100k	MovieLens-1m
#ratings	100 000	1 000 209
#users	943	6040
#movies	1682	3706
Average rating	3.53	3.58
Ratings standard deviation	1.13	1.12

Table 3.1: The datasets' details.

The user rating is represented by a tuple of 4 values — `<user, item, rating, timestamp>` (Table 3.2). Moreover, there is an auxiliary dataset

(Table 3.3) containing information about the movies themselves. It can be further used to make embeddings.

user_id	movie_id	rating	timestamp
1	1193	5	978300760
454	1036	1	976287910
978	2997	4	975107102
1908	194	3	974873117
2969	2581	4	982860538

Table 3.2: The sample of users' ratings from MovieLens-100k dataset.

movie_id	name	genres
1	Toy Story (1995)	Animation|Children's|Comedy
2	Jumanji (1995)	Adventure|Children's|Fantasy
3	Grumpier Old Men (1995)	Comedy|Romance
4	Waiting to Exhale (1995)	Comedy|Drama
5	Father of the Bride Part II (1995)	Comedy

Table 3.3: The sample of movies' data from the auxiliary dataset.

3.2 Data Preparation

- The datasets were used in their original form; no films nor movies were filtered out.

- We have constructed a user-item interaction matrix, also known as *pivot table*, where each row represents a user profile (Figure 3.4). A user profile is a sparse vector, where each nonzero element is a rating from 1 to 5. Non-zero values mean that a user did not interact with the corresponding items.

- In order to compare different items, item embeddings were used. The embedding is a sparse one-hot encoded vector, where each value represents some feature. The resulting embeddings were represented by 28675-valued vectors and were constructed using the ontology-based approach. A detailed description of that approach can be found in [23].

user_id \ item_id	1	2	...	n
1	0	1	...	1
2	0	0	...	0
3	0	1	...	0
4	1	0	...	1

Table 3.4: The sample of user-item interaction matrix for 4 users and n items.

3.3 Modeling

In recommendation systems, there are plenty of data mining algorithms that can be used, such as KNN, Decision trees, SVM (Support Vector Machine), Neural networks, etc. They can be used in previously discussed RS approaches.

3.3.1 K-Nearest Neighbours

K-Nearest Neighbours (KNN) is an algorithm that considers user similarities in order to produce recommendations. It works in the following 3 steps: (1) finding k most similar users to the target user, (2) finding the items these "nearest neighbours" liked or interacted with, and then (3) recommending it to the target user. We can employ any similarity metric (l2 norm, cosine, etc.) to compare the users. KNN algorithm is not time efficient, since it requires to compute distances to all other users, what is asymptotically equal to $O(n)$. KNN can be applied in both content-based and user-based recommendation systems.

We have chosen a *popularity-stratified* variation of the KNN algorithm with cosine similarity and voting for our experiments. It considers the popularity of the items, giving more preference to unpopular ones. Such an approach intuitively leads to better diversification. The rank function is given as follows:

$$rank(u,i) = \frac{\sum_{\hat{u} \in N_u^k} sim(u,\hat{u}) \cdot (r_{\hat{u},i} - \bar{r}_{\hat{u}})}{(\sum_{\hat{u} \in U} (r_{\hat{u},i} - \bar{r}_{\hat{u}}))^{\beta}} \tag{3.1}$$

where N_u^k denotes a set of the k nearest neighbours of user u, U denotes a set of all users, and β denotes a long-tail biasing parameter proposed by [24].

3.3.2 Matrix factorization

Generally, the user-item interaction matrix is huge and sparse. We can decompose that matrix into the product of lower dimensionality matrices by using the matrix factorization (MF) technique. It allows us to further work with smaller matrices and make all computations faster. There are different MF algorithms, such as Truncated SVD or PCA.

3.3.2.1 Truncated SVD

Singular value decomposition (SVD) is a matrix factorization technique commonly used for producing low-rank approximations. Given a matrix $X \in \mathbb{R}^{m \times n}$ with $rank(A) = r$.

$$X = USV^T, \tag{3.2}$$

where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices, with their columns being the eigenvectors of XX^T and X^TX, respectively, $S \in \mathbb{R}^{m \times n}$ is a diagonal matrix with r nonzero elements, which are singular values of X. The diagonal r elements $(\sigma_1, \sigma_2, ..., \sigma_r)$ of S are sorted in a descending order, i.e., $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r > 0$.

In recommendation systems, a variant of SVD, *Truncated SVD*, is usually used. It essentially computes only k largest singular values, where k is a user-specified parameter. Truncated SVD applied to the training data X produces a closest rank-k approximation X. Mathematically, it can be written down as follows:

$$X \approx X_k = U_k S_k V_k^T, \tag{3.3}$$

where $U_k \in \mathbb{R}^{m \times k}$, $S_k \in \mathbb{R}^{k \times k}$, $V_k \in \mathbb{R}^{k \times n}$ are matrices. Matrix $U_k S_k^T$ represents the transformed training data with k features.

3.3.3 Neural networks

Restricted Boltzmann machine (RBM) is essentially a stochastic neural network with two layers, called "visible" and "hidden," which form a bipartite graph. The visible layer is a known user profile, and the hidden layer is the latent factors that we want to learn. The learning process is unsupervised and involves using the Contrastive Divergence technique.

AutoRec is a novel algorithm proposed by [25], that is based on the autoencoder paradigm. AutoRec is a fast and memory-efficient algorithm, with a less number of parameters in comparison with RBM and outperforming it. The architecture consists of 3 parts: encoder, latent space, and decoder. The input, represented by a user profile, is mapped to the latent space, and then the decoder tries to reproduce the input. The hidden layer consists of latent factors, which size is determined by the hidden layer's size. Latent factors are implicit features, which are determined automatically from the input data. In the case of a movie dataset, the latent factor may represent user preferences, such as whether a user is a fan of some specific movie genre.

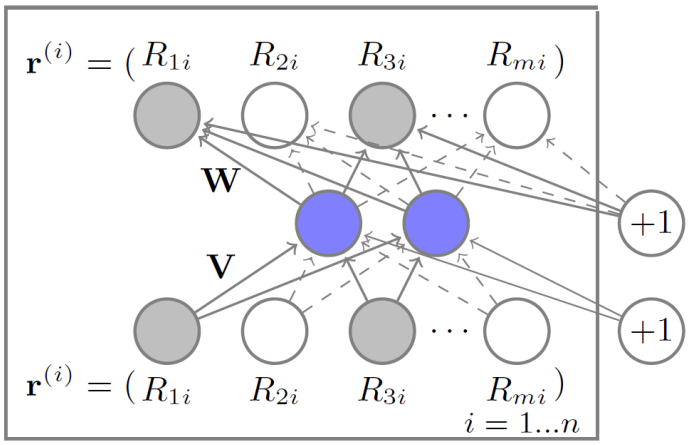

Figure 3.1: AutoRec architecture. [25]

3.3.4 Ensemble

In machine learning, the ensemble method is a technique that presumes the usage of multiple models to obtain better predictive results. Our ensemble consists of 3 models: KNN, Matrix Factorization and AutoRec. The outputs of each model are combined into a list. Then, the list is sorted by a desired metric, like recall, coverage, serendipity. And, finally, the selection of k best items forms the final output. It is also possible to order items by several different metrics. For example, suppose we want to optimize two metrics, recall and serendipity, simultaneously. In that case, we select the first half items sorted by recall and the second half items sorted by serendipity.

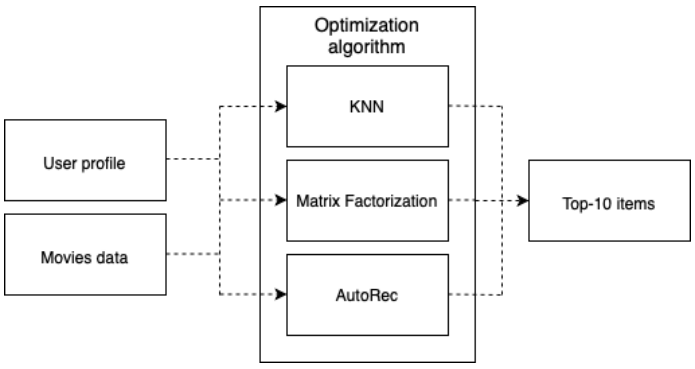

Figure 3.2: 3-models ensemble diagram.

3.3.5 Meta-learning

In our experiments, we calculate the following meta-data for every dataset:

1. Average rating per item.

2. Ratings standard deviation.

3. Number of features.

4. Average features per item.

5. Features standard deviation.

In addition, we save the following meta-knowledge:

1. Dataset name.

2. Model type.

3. Best parameters.

Consequently, we used meta-data to better initialize the optimization algorithms. If meta-data for two different datasets is similar, we can consider starting the searching process using specific hyperparameters.

3.3.6 Hyperparameter optimization

In our trials, we have employed grid search and NSGA2 methods to fine-tune our algorithms, which were described in the previous chapter. NSGA2 is an evolutionary algorithm and thus requires a fitness function to evaluate the quality of the individuals. The implementation of the fitness function for NSGA2 algorithm is described in Algorithm 2.

Algorithm 2: Fitness function for NSGA2

```
R ← ∅;
initialize models;
foreach user do
    foreach model do
        r ← model.recommend(user);
        R(user) ← R(user) ⋃ r;
    end
    R(user) ← 10 random items from R(user);
end
evaluated objective functions of R;
return calculated scores;
```

3.4 Evaluation methodology

Given a dataset, the typical way of estimating the quality of the recommendation system is to use *split validation* [26]. The validation stage plays a crucial role in developing a successful model.

Split validation works as follows. Let $\mathcal{D}$ denote the whole dataset. Then a random subset $\mathcal{D}_{\text{train}} \in \mathcal{D}$, called the *training set*, is selected and used to train algorithms. The rest $\mathcal{D} \setminus \mathcal{D}_{\text{train}}$ is usually divided into the *test set* $\mathcal{D}_{\text{test}}$ and *validation set* $\mathcal{D}_{\text{val}}$ such that $\mathcal{D}_{\text{test}} \cup \mathcal{D}_{\text{val}} = \mathcal{D} \setminus \mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}} \cap \mathcal{D}_{\text{val}} = \emptyset$. Usually, validation set is used to optimize hyperparameters and test set is used to evaluate the resulting model. The motivation for introducing $\mathcal{D}_{\text{val}}$ is the fact that the split is done randomly, therefore there is a risk of overfitting through hyperparameter search towards the particular split [26].

For the trials, we split the data into training and test sets, keeping 90% of the data for the training set and 10% for the test set. We did not employ a validation set due to the relatively small dataset size and computational reasons.

The recall, catalogue coverage, and serendipity were measured as follows. For each user from the test set, only one *control item* was left. Let R_u denote top-N recommendations generated by the algorithm. Then, if R_u contains a *control item*, we set the recall of R_u to 1 or 0 otherwise. We calculate the sum of recall for all R_u and divide it by the number of users in the test set. The resulting value represents the final recall of the algorithm. Catalogue coverage and serendipity were measured straightforwardly.

Chapter 4

Experiments

In this chapter, we will discuss the experiments that we have conducted. In the beginning, we will describe the experimental setup, which includes algorithms and metrics configuration. Then we will describe the obtained results and discuss the conclusions we made out.

4.1 Offline Experiments Setup

We have conducted experiments employing three algorithms we have discussed in section 3.3: popularity-stratified KNN, Matrix Factorization (Truncated SVD), and AutoRec.

The essential points are formulated as a list:

- The random seed for all methods and algorithms was set to 42.
- The first set of experiments was conducted with β set to 0 and 1, whereas the second and third sets of experiments were conducted using $\beta = 1$.
- Serendipity metric, defined as (2.11), requires predefined weights for every auxiliary metric. Parameters α, β, γ were set to 0.4, 0.3, 0.3, respectively.
- We selected a simple user-based 3-NN model as a primitive model since unexpectedness measuring demands it.
- MongoDB Atlas Database[7] represents a meta-storage. Meta-knowledge are stored there.

[7] `https://www.mongodb.com/cloud/atlas`

Algorithm	Hyperparameters
Popularity-stratified KNN	The number of neighbours
Matrix Factorization (Truncated SVD)	The number of components
AutoRec	Hidden layer size

Table 4.1: Algorithms overview.

4.2 Results

The first experiments are done using grid search to test different hyperparameters on KNN, MF, and AutoRec algorithms.

- Tables 4.4 and 4.5 show comprehensive grid search results.
- Figures 4.1 and 4.2 give an overview of how models behave with different hyperparameters. Each row of the graph corresponds to the specific model and each column corresponds to the specific metric.
- Figures 4.3 and 4.4 show a correlation between recall and serendipity.
- Figures 4.5 and 4.6 show the same results, but between coverage and serendipity.

The results for KNN model in Figures 4.1 and 4.2 show us how β impacts the metrics. As expected, serendipity and catalogue coverage are higher for $\beta = 1$ since unwanted items are more likely to be recommended. Nevertheless, recall drastically decreases. In case $\beta = 1$, the situation is inversed - recall is high, but serendipity and catalogue coverage is low.

There is a slightly negative correlation between serendipity and recall that can be observed from Figures 4.4 and 4.3.

In Figure 4.6, we can notice that there is a correlation between serendipity and catalogue coverage. It is arising from the fact that the serendipitous RS promotes novel and unexpected items. It intuitively leads to better diversification and catalogue coverage improvement. In the similar graph 4.5, which was evaluated on MovieLens-100k, the same dependency can be observed, but it is less pronounced.

Looking at Tables 4.4 and 4.5 we can note that hyperparameter and β heavily impact the training and evaluating time for the *KNN* model. The number of neighbors may result in five times worse efficiency, and β may result in more than two times worse efficiency. With a growing k, recall, catalogue coverage, and serendipity are decreasing due to the possible overfitting. Thus we would choose *five* as the optimal value of k.

In contrast, *MF* and *AutoRec* models do not suffer from the problem when hyperparameters affect the overall training and evaluating time. The time is nearly the same for all hyperparameters that we have tested.

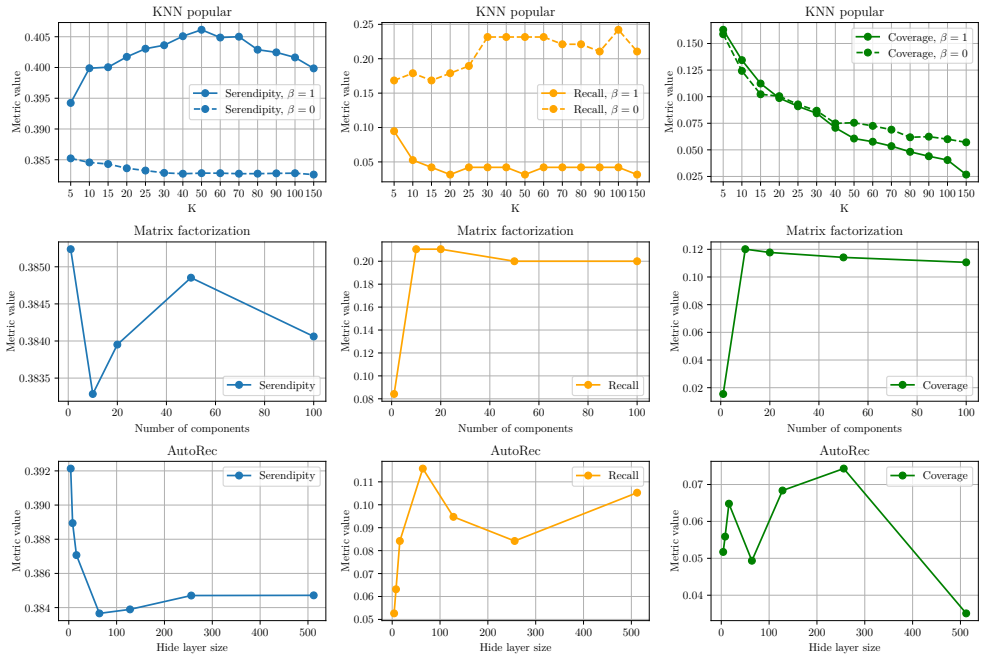

Figure 4.1: Models overview evaluated on MovieLens-100k. Each row corresponds to the specific model.

The MF algorithm performs the best with 20-100 components and provides nearly the same performance.

The AutoRec algorithm benefits the most from 64-256 hidden layer sizes. For the low number of neurons in the hidden layer, the metrics are decreasing, while the high number results in low catalogue coverage.

The second set of experiments were performed with the NSGA2 algorithm optimizing the proposed ensemble method. The serendipity and recall metrics were chosen as objective functions.

- The resulting individuals after five generations of the NSGA2 algorithm are shown in Table 4.2. The population size was set to 7. The fitness function was implemented as in Algorithm 2. The total spent time is 26.5 hours.

The third and the last set of experiments were conducted with the meta-learning technique. The previous experiment is related to this one. The NSGA2 algorithm optimized the ensemble method, and meta-learning was used in the population initialization.

- Table 4.3 shows how meta-learning impacts the metrics. Meta-data of MovieLens-100k were used in order to better initialize the population

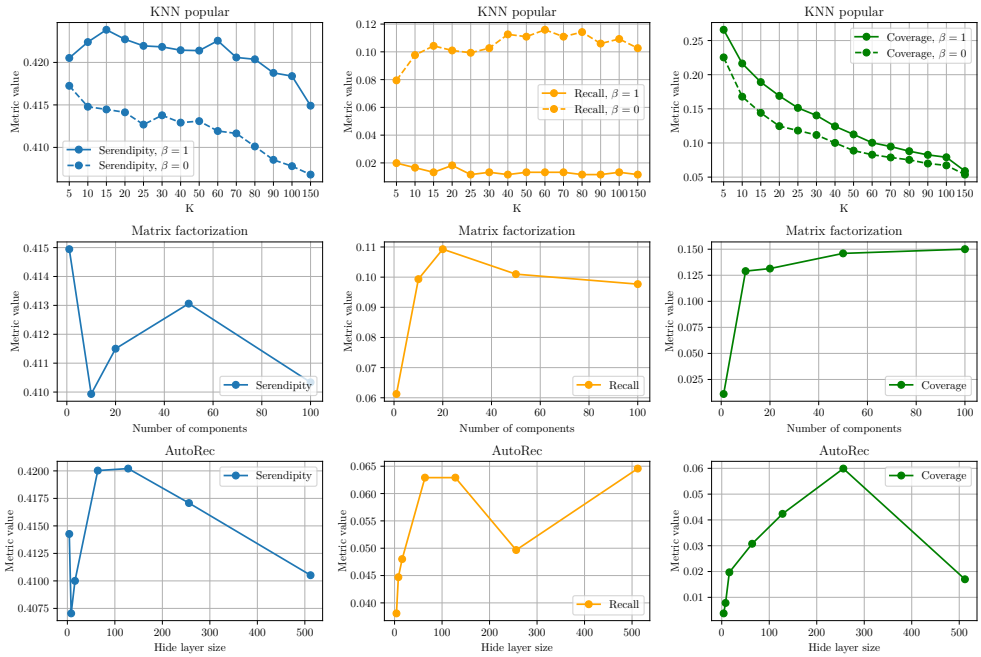

Figure 4.2: Models overview evaluated on MovieLens-1m. Each row corresponds to the specific model.

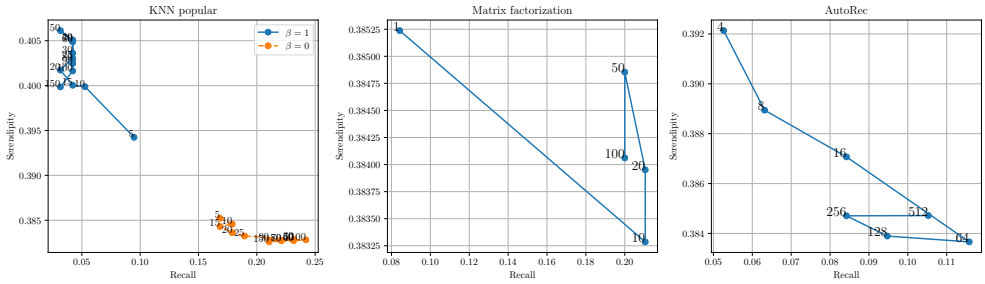

Figure 4.3: Recall versus serendipity graphs evaluated on MovieLens-100k.

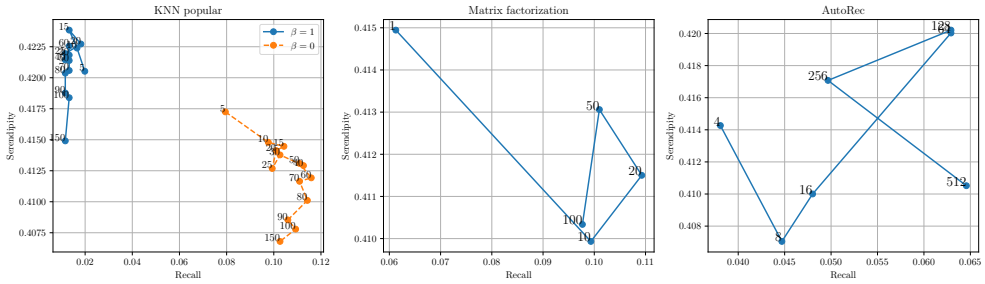

Figure 4.4: Recall versus serendipity graphs evaluated on MovieLens-1m.

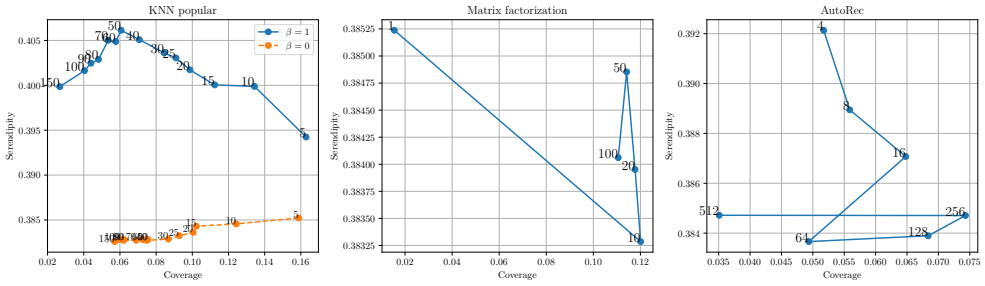

Figure 4.5: Catalogue coverage versus serendipity graphs evaluated on MovieLens-100k.

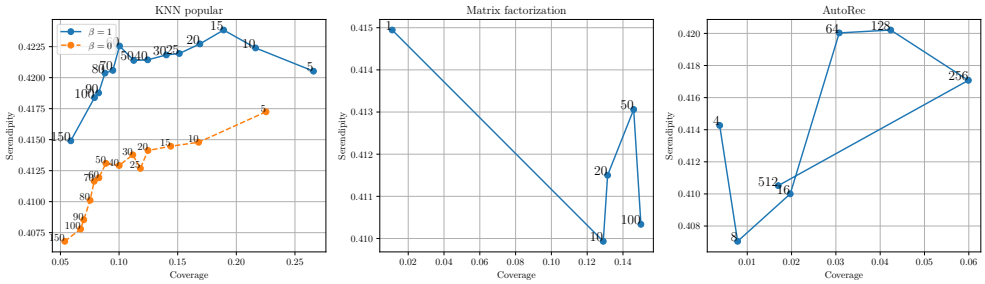

Figure 4.6: Catalogue coverage versus serendipity graphs evaluated on MovieLens-1m.

KNN	MF	AutoRec	Serednipity	Recall	Coverage
61	22	96	0.420	0.063	0.111
59	84	72	0.420	0.052	0.120
56	21	152	0.417	0.065	0.116
59	90	128	0.417	0.067	0.117
61	15	128	0.420	0.050	0.113

Table 4.2: The best individuals of the ensemble method optimized by NSGA2 with 5 iterations and population size 7. Evaluated on MovieLens-1m.

for the NSGA2 algorithm. The recall for the meta-learning approach reaches 0.068, while the usual approach reaches just 0.058.

4.3 Discussion

One of the conclusions of the first experiment is that the KNN algorithm with a moderate k and β set to 0 performs better than any other model if we do not consider training and evaluating time. When β is equal to 1, serendipity is slightly increasing, but recall becomes extremely low. The fastest algorithm

KNN	MF	AutoRec	Seredipity	Recall	Coverage
60	10	168	0.421	0.055	0.110
70	10	168	0.420	0.058	0.111
73	16	232	0.419	0.068	0.108
67	19	48	0.421	0.050	0.110

(a) Meta-learning combined with random initialization.

KNN	MF	AutoRec	Seredipity	Recall	Coverage
35	30	320	0.419	0.055	0.132
67	22	464	0.420	0.050	0.110
66	17	440	0.421	0.040	0.107
31	13	104	0.418	0.058	0.132

(b) Random initialization.

Table 4.3: Table (a) contains the results of an approach with an involved meta-learning and Table (b) contains the results of a common approach. Evaluated on MovieLens-1m.

is AutoRec, which is up to 13 times faster than KNN and up to 5 times faster than MF. However, AutoRec's performance is quite questionable. MF is an intermediate option, which offers good performance and adequate time costs.

Based on the second experiment, we came to the conclusion that the proposed ensemble method performs more or less the same as the MF algorithm. However, the overall training time numbers in dozens, making the ensemble method more complex to train and evaluate.

From the third experiment, we can conclude that the meta-learning approach has utility when the time costs are crucial. It improves the results when the total training time is limited.

Model	Serendipity	Recall	Coverage	Time[a]
KNN, $\beta = 1$ (k=5)	0.395	0.053	0.148	116
KNN, $\beta = 1$ (k=10)	0.400	0.042	0.124	130
KNN, $\beta = 1$ (k=15)	0.403	0.032	0.106	141
KNN, $\beta = 1$ (k=20)	0.407	0.032	0.093	150
KNN, $\beta = 1$ (k=25)	0.408	0.042	0.089	160
KNN, $\beta = 1$ (k=30)	0.408	0.042	0.081	169
KNN, $\beta = 1$ (k=40)	0.408	0.032	0.069	185
KNN, $\beta = 1$ (k=50)	0.409	0.032	0.056	199
KNN, $\beta = 1$ (k=60)	0.408	0.032	0.054	212
KNN, $\beta = 1$ (k=70)	0.407	0.042	0.049	226
KNN, $\beta = 1$ (k=80)	0.405	0.042	0.043	239
KNN, $\beta = 1$ (k=90)	0.404	0.042	0.042	251
KNN, $\beta = 1$ (k=100)	0.404	0.042	0.038	263
KNN, $\beta = 1$ (k=150)	0.405	0.032	0.030	319
KNN, $\beta = 0$ (k=5)	0.385	0.168	0.159	117
KNN, $\beta = 0$ (k=10)	0.385	0.179	0.124	133
KNN, $\beta = 0$ (k=15)	0.384	0.168	0.102	146
KNN, $\beta = 0$ (k=20)	0.384	0.179	0.100	156
KNN, $\beta = 0$ (k=25)	0.383	0.189	0.093	167
KNN, $\beta = 0$ (k=30)	0.383	0.232	0.087	176
KNN, $\beta = 0$ (k=40)	0.383	0.232	0.075	191
KNN, $\beta = 0$ (k=50)	0.383	0.232	0.076	206
KNN, $\beta = 0$ (k=60)	0.383	0.232	0.073	219
KNN, $\beta = 0$ (k=70)	0.383	0.221	0.069	234
KNN, $\beta = 0$ (k=80)	0.383	0.221	0.062	246
KNN, $\beta = 0$ (k=90)	0.383	0.211	0.062	256
KNN, $\beta = 0$ (k=100)	0.383	0.242	0.060	269
KNN, $\beta = 0$ (k=150)	0.383	0.211	0.057	323
MF (SVD, n_comp=1)	0.385	0.084	0.015	176
MF (SVD, n_comp=10)	0.383	0.211	0.120	142
MF (SVD, n_comp=20)	0.384	0.211	0.118	142
MF (SVD, n_comp=50)	0.385	0.200	0.114	143
MF (SVD, n_comp=100)	0.384	0.200	0.111	145
AutoRec (hide_layer=4)	0.394	0.074	0.052	55
AutoRec (hide_layer=8)	0.395	0.105	0.051	53
AutoRec (hide_layer=16)	0.388	0.074	0.033	55
AutoRec (hide_layer=64)	0.384	0.095	0.058	58
AutoRec (hide_layer=128)	0.383	0.053	0.062	61
AutoRec (hide_layer=256)	0.384	0.063	0.064	59
AutoRec (hide_layer=512)	0.385	0.095	0.038	57

[a]Training and evaluating time in seconds.

Table 4.4: Models performance on MovieLens-100k dataset.

Model	Serendipity	Recall	Coverage	Time[a]
KNN, $\beta = 1$ (k=5)	0.423	0.020	0.262	1374
KNN, $\beta = 1$ (k=10)	0.422	0.017	0.212	2259
KNN, $\beta = 1$ (k=15)	0.423	0.013	0.184	2475
KNN, $\beta = 1$ (k=20)	0.423	0.018	0.164	2766
KNN, $\beta = 1$ (k=25)	0.422	0.012	0.144	3029
KNN, $\beta = 1$ (k=30)	0.422	0.013	0.134	3169
KNN, $\beta = 1$ (k=40)	0.422	0.012	0.119	3549
KNN, $\beta = 1$ (k=50)	0.422	0.012	0.110	3949
KNN, $\beta = 1$ (k=60)	0.423	0.013	0.097	4239
KNN, $\beta = 1$ (k=70)	0.421	0.013	0.091	4924
KNN, $\beta = 1$ (k=80)	0.420	0.013	0.086	5498
KNN, $\beta = 1$ (k=90)	0.419	0.013	0.082	5561
KNN, $\beta = 1$ (k=100)	0.418	0.013	0.079	5820
KNN, $\beta = 1$ (k=150)	0.415	0.012	0.058	7185
KNN, $\beta = 0$ (k=5)	0.417	0.079	0.225	615
KNN, $\beta = 0$ (k=10)	0.415	0.098	0.168	817
KNN, $\beta = 0$ (k=15)	0.414	0.104	0.144	963
KNN, $\beta = 0$ (k=20)	0.414	0.101	0.125	1101
KNN, $\beta = 0$ (k=25)	0.413	0.099	0.118	1220
KNN, $\beta = 0$ (k=30)	0.414	0.103	0.112	1336
KNN, $\beta = 0$ (k=40)	0.413	0.113	0.100	1537
KNN, $\beta = 0$ (k=50)	0.413	0.111	0.089	1729
KNN, $\beta = 0$ (k=60)	0.412	0.116	0.083	1899
KNN, $\beta = 0$ (k=70)	0.412	0.111	0.079	2069
KNN, $\beta = 0$ (k=80)	0.410	0.114	0.075	2230
KNN, $\beta = 0$ (k=90)	0.409	0.106	0.070	2380
KNN, $\beta = 0$ (k=100)	0.408	0.109	0.067	2532
KNN, $\beta = 0$ (k=150)	0.407	0.103	0.054	3227
MF (SVD, n_comp=1)	0.415	0.061	0.011	2556
MF (SVD, n_comp=10)	0.410	0.099	0.129	2453
MF (SVD, n_comp=20)	0.412	0.109	0.131	2071
MF (SVD, n_comp=50)	0.413	0.101	0.146	2248
MF (SVD, n_comp=100)	0.410	0.098	0.150	2166
AutoRec (hide_layer=4)	0.421	0.036	0.028	529
AutoRec (hide_layer=8)	0.406	0.045	0.006	567
AutoRec (hide_layer=16)	0.411	0.060	0.009	584
AutoRec (hide_layer=64)	0.415	0.061	0.028	628
AutoRec (hide_layer=128)	0.418	0.068	0.048	692
AutoRec (hide_layer=256)	0.418	0.055	0.062	623
AutoRec (hide_layer=512)	0.417	0.063	0.019	650

[a] Training and evaluating time in seconds.

Table 4.5: Models performance on MovieLens-1m dataset.

Chapter 5

Conclusion

In this final chapter, we will conclude the whole thesis, including the opportunities for future research.

5.1 Summary

In this thesis, we addressed the hyperparameter optimization problem in recommendation systems employing brute-force and evolutionary multi-objective approaches. We tested the state-of-the-art algorithms on publicly available academic MovieLens datasets. We presented the alternative definition of serendipity based on previous studies. We proposed an ensemble model, which eventually performed almost as a Matrix Factorization algorithm. We proposed a meta-learning approach which resulted in better performance for time-bounded optimization. We studied the relationship between the evaluated metrics and concluded that serendipity negatively correlates with recall, resulting in a recall-serendipity trade-off.

5.2 Future work

While working on this thesis, the author came up with several ideas to be explored:

- Evaluating approaches using more extensive datasets, e.g., MovieLens-20M or Netflix Prize.
- Developing a data processing framework designed for recommendation systems.

Bibliography

[1] Candillier, L.; Jack, K.; et al. State-of-the-art recommender systems. In *Collaborative and Social Information Retrieval and Access: Techniques for Improved User Modeling*, Orange Labs Lannion, France: IGI Global, 2009, ISBN 9781605663067, pp. 1–22.

[2] Isinkaye, F.; Folajimi, Y.; et al. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, volume 16, no. 3, 2015: pp. 261–273, ISSN 1110-8665.

[3] IBM. What is Machine Learning? [online]. 15.07.2020 [cit. 20.04.2021]. Available from: `https://www.ibm.com/cloud/learn/machine-learning`

[4] He, X.; Zhao, K.; et al. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, volume 212, Jan 2021: p. 106622, ISSN 0950-7051.

[5] Hutter, F.; Kotthoff, L.; et al. (editors). *Automated Machine Learning: Methods, Systems, Challenges*. Switzerland: Springer Nature, 2019, ISBN 978-3-030-05318-5.

[6] Kordík, P.; Koutník, J.; et al. Meta-learning approach to neural network optimization. *Neural networks : the official journal of the International Neural Network Society*, volume 23, 02 2010: pp. 568–82.

[7] Ying, C.; Klein, A.; et al. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, PMLR, 2019, pp. 7105–7114.

[8] Pilario, K. E. S.; Cao, Y.; et al. A Kernel Design Approach to Improve Kernel Subspace Identification. *IEEE Transactions on Industrial Electronics*, volume 68, no. 7, 2021: pp. 6171–6180.

[9] Bergstra, J. S.; Bardenet, R.; et al. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems 24*, edited by J. Shawe-Taylor; R. S. Zemel; P. L. Bartlett; F. Pereira; K. Q. Weinberger, Curran Associates, Inc., 2011, pp. 2546–2554.

[10] Shani, G.; Gunawardana, A. Evaluating recommendation systems. In *Recommender systems handbook*, Springer, 2011, pp. 257–297.

[11] Vargas, S.; Castells, P. Rank and Relevance in Novelty and Diversity Metrics for Recommender Systems. In *Proceedings of the fifth ACM conference on Recommender systems*, New York, NY, USA: Association for Computing Machinery, 2011, ISBN 9781450306836, p. 109–116.

[12] Kotkov, D.; Konstan, J. A.; et al. Investigating Serendipity in Recommender Systems Based on Real User Feedback. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, New York, NY, USA: Association for Computing Machinery, 2018, ISBN 9781450351911, p. 1341–1350.

[13] Murakami, T.; Mori, K.; et al. Metrics for Evaluating the Serendipity of Recommendation Lists. In *New Frontiers in Artificial Intelligence*, edited by K. Satoh; A. Inokuchi; K. Nagao; T. Kawamura, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, ISBN 978-3-540-78197-4, pp. 40–46.

[14] Ge, M.; Delgado-Battenfeld, C.; et al. Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *Proceedings of the fourth ACM conference on Recommender systems*, 2010, pp. 257–260.

[15] Deb, K. Multi-objective optimisation using evolutionary algorithms: an introduction. In *Multi-objective evolutionary optimisation for product design and manufacturing*, Springer, 2011, pp. 3–34.

[16] Blank, J.; Deb, K. pymoo: Multi-objective optimization in python. *IEEE Access*, volume 8, 2020: pp. 89497–89509.

[17] Deb, K.; Pratap, A.; et al. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, volume 6, no. 2, 2002: pp. 182–197.

[18] Kok, J.; Gonzalez, F.; et al. An FPGA-based approach to multi-objective evolutionary algorithm for multi-disciplinary design optimisation. In *Proceedings of the International Conference on Evolutionary and Deterministic Methods for Design, Optimization and Control with Applications to Industrial and Societal Problems (Eurogen 2011)*, CIRA-Italian Aerospace Research Centre, 2011, pp. 1–10.

[19] Wang, T.-H.; Hu, X.; et al. AutoRec: An Automated Recommender System. In *Fourteenth ACM Conference on Recommender Systems*, 2020, pp. 582–584.

[20] Anand, R.; Beel, J. Auto-Surprise: An Automated Recommender-System (AutoRecSys) Library with Tree of Parzens Estimator (TPE) Optimization. *Fourteenth ACM Conference on Recommender Systems*, Sep 2020.

[21] Bergstra, J.; Komer, B.; et al. Hyperopt: A Python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, volume 8, 07 2015: p. 014008.

[22] Harper, F. M.; Konstan, J. A. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.*, volume 5, no. 4, Dec. 2015, ISSN 2160-6455.

[23] Kuznetsov, S. *Ontologies in Recommender Systems*. Dissertation thesis, České vysoké učení technické v Praze. Fakulta informačních technologií, 2020.

[24] Steck, H. Item Popularity and Recommendation Accuracy. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, New York, NY, USA: Association for Computing Machinery, 2011, ISBN 9781450306836, p. 125–132.

[25] Sedhain, S.; Menon, A. K.; et al. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th international conference on World Wide Web*, 2015, pp. 111–112.

[26] Tomáš, Ř. *Manipulating the Capacity of Recommendation Models in Recall-Coverage Optimization*. Dissertation thesis, České vysoké učení technické v Praze. Vypočetní a informační centrum., 2019.

Appendix A

Acronyms

RS Recommendation system

CF Collaborative filtering

CBF Content-based filtering

ML Machine Learning

EA Evolutionary algorithm

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `thesis.pdf` ..... the thesis text in PDF format
- `src` ..... the directory of source codes
  - `experiments` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis