Praca dyplomowa magisterska

Jacek Głodek

Recommender System for a Knowledge Base

Opiekun pracy:
prof. nzw. dr hab. Piotr Gawrysiak

Ocena ............................

..................................

Podpis Przewodniczącego
Komisji Egzaminu Dyplomowego
**Kierunek:** Informatyka  

**Specjalność:** Inżynieria Systemów Informatycznych

**Data urodzenia:** 1989.02.24  

**Data rozpoczęcia studiów:** 2008.10.01

---

**Życiorys**


………………………………………

podpis studenta

---

**Egzamin dyplomowy**

Złożył egzamin dyplomowy w dn. …………………………………………………

Z wynikiem ………………………………………………………………………

Ogólny wynik studiów ……………………………………………………………

Dodatkowe wnioski i uwagi Komisji ………………………………………………

…………………………………………………………………………………………
Abstract

This thesis describes recommender systems, their classification, mechanisms used for recommendation and implementation of an recommender system. The purpose of this thesis was extending MediaWiki encyclopedic knowledge base with recommendation features.

Key words: recommender systems, human-computer interaction, expert knowledge, information filtering, content-based recommendation, collaborative filtering, document correlation, MediaWiki, Ruby

Streszczenie

Tytuł pracy: System rekomendacyjny dla bazy wiedzy.

Praca ta opisuje systemy rekomendacyjne, klasyfikację systemów rekomendacyjnych, mechanizmy w nich wykorzystywane oraz implementację jednego systemu rekomendacyjnego. Celem tej pracy było rozwinięcie encyklopedycznej bazy wiedzy MediaWiki o funkcjonalność rekomendacji artykułów użytkownikom, ze względu na ich zainteresowania.

Słowa kluczowe: systemy rekomendacyjne, systemy eksperckie, filtrowanie informacji, korelacja danych tekstowych, wiedza ekspercka, MediaWiki, Ruby, treść
Contents

1. Prologue - The Recommendation Systems .................................................. 1
   1.1. Computer recommendation systems .................................................. 1
   1.2. Recommendation ............................................................................... 2

2. Classification of recommendation systems ................................................. 4
   2.1. Classification by subject of recommendation ........................................ 4
      2.1.1. Content recommendation systems .............................................. 5
      2.1.2. Action recommender systems ..................................................... 7
      2.1.3. Strategy recommendation systems .............................................. 8
   2.2. Classification by environmental constraints ......................................... 9
   2.3. Classification by prediction knowledge source ..................................... 11
      2.3.1. Recommendation systems based on expert knowledge ................. 12
      2.3.2. Expert knowledge based recommendation - Rules and Ontologies ......................................................................................... 13
      2.3.3. Expert knowledge based recommendation - Mathematical model-based and heuristic-based .............................................................. 14
      2.3.4. Recommendation systems based on computer learning and data analysis ................................................................. 16
      2.3.5. Adaptive recommendation models .............................................. 17
      2.3.6. Artificial intelligence based recommendation models ................. 18

3. Recommendation mechanisms ................................................................. 19
   3.1. User-wise mechanisms ....................................................................... 19
   3.2. Content-wise mechanisms .................................................................. 20
   3.3. Social-wise mechanisms .................................................................... 23
   3.4. Social-graph and neighourhood-based mechanisms ............................ 24
   3.5. Mathematical approach ..................................................................... 24

4. Crowdsourced Knowledge Base for Fast Information Search .................... 26
   4.1. Problem description ........................................................................... 26
   4.2. MediaWiki system as a solution .......................................................... 27
      4.2.1. Crowdsourcing and user management in MediaWiki .................. 28
      4.2.2. MediaWiki as Content Management System ............................. 28
      4.2.3. MediaWiki Scalability and Technology ...................................... 29
      4.2.4. Flaws of MediaWiki as a solution .............................................. 29
1. Prologue - The Recommendation Systems

Computer systems grow in complexity. Due to accessibility of computer hardware and software, computers are used by broader audience, that grows every day\(^1\). Therefore, task of supporting human-computer interaction is important\(^2\). One of the ways to support usage of the computer, are recommender systems\(^3\). These systems give user advices, that help him use the computer.

The purpose of this thesis is to present basic concept of recommendation systems and describe implementation of recommender system supporting use of a knowledge base. I present classification of recommender systems by various measures. In this thesis I focus on general ideas of recommendation. I don’t explain technologies that implement it or any implementational materials, as it could cloud the general view of recommender systems\(^4\).

In first chapter of this thesis I give a brief description of recommender systems. The second chapter describes classification of recommender systems. Third chapter describes mechanisms that can be used for recommendation. The rest of the thesis presents my implementation of content recommendation system for a knowledge base, based on MediaWiki content management system.

I assume reader has basic knowledge of computer systems, mathematics, data and text-mining methods. I hope that reader will understand basic idea of recommendation system and avoid being affected by pigeonholed categorization of information filtering systems. I use broad definition of recommender system in order to highlight that.

1.1. Computer recommendation systems

Computer recommendation systems are often defined as a subclass of information filtering systems\(^5\). In this book I use broader understanding of recommendation systems, which is:

---

4. McClamrock, “Marr’s three levels: A re-evaluation”
1.2. Recommendation

Computer recommendation system is auxiliary computer program or set of computer system’s features that support user in usage of the system, by providing advice.

Given this definition the idea of recommendation system is broaden to: information filtering system, decision support systems, wizards, accessible user manuals, tutorials, usage tips, etc.

This definition also implies that supported system and it’s actions are not necessarily computer ones. Also, the decisive element is not considered as a part of the recommendation system.

Computer systems main purpose is to provide some functionality to the user. Given that, we can define:

**Recommendation system’s main functionality is recommending user with the most useful actions. Optionally it should explain the choice that was made.**

That requires the system to consider current context of the user actions, which can be described as:

— user profile,
— content and actions of the system,
— environmental context.

User profile is a set of informations about the user. System tracks what user does, what are his interests and goals, and tries to advise him with relevant informations.

Content and actions is what the system operates on. Considering content and actions, the system should understand what actions is user authorized to perform, or physically able to perform. How different items of content are correlated and what do they present.

Environmental context is broad description on every static or dynamic property of the system itself. It can be sensoric information, situation within the system, user profiles of all the users, track of most popular actions in system, etc..

1.2. Recommendation

Recommendation can be described as act of recommending. To recommend something means to present something as worthy of acceptance or trial. In computers domain, computers recommend interactions to the user. It is important to notice, that recommendation is not equal acting on behalf of the user. The system’s user is responsible for decisions. Recommendations can have various properties that characterize them. Some of these properties are:

---

6 “Merriam-Webster’s Medical Dictionary”.
7 Baecker, *Readings in Human-Computer Interaction: toward the year 2000*.
1.2. Recommendation

**User Preference** - Recommendations are made using knowledge about user's interest, habits and goals.

**Prediction Accuracy** - Accurate recommendations are made, when system properly predicted results of following the recommendation.

**Coverage** - Coverage is a degree to which recommendations cover available items and actions in the system.

**Confidence** - Recommendation system can provide information, about how much is he confident with recommendation choice.

**Trust** - User's trust in system recommendations. Ex. if system explains his recommendation decisions or decisions are reasonable, user trusts the recommendations more.

**Novelty** - Novel recommendations are recommendations for items that user did not know about.

**Serendipity** - Measure of how surprising the successful recommendations are.

**Diversity** - Diverse recommendations offer broad choice for the user.

**Utility** - Measure of how is the recommendation useful for the user.

**Risk** - Recommender system can embrace and explain risks of following recommendations to the user.

**Robustness** - It is stability of the system in the presence of fake information or abuse.

**Privacy** - Although, recommender systems operate on user's preferences that are disclosed willingly, the user usually wants his preferences to be private.

**Adaptivity** - When systems content and environment are dynamic, the recommendations should stay accurate, the system should adapt.

**Scalability** - The system should be able to handle growing amount of data.

Measuring and managing recommendation system properties we can provide systems that are more usable.
2. Classification of recommendation systems

To understand what is the domain of recommendation system’s research, it is convenient to classify them by different means. Following sections will explain different types of recommendation systems. Various approaches to recommendation systems differ in:

- subject of recommendation,
- environment and constraints of recommendation,
- data acquisition methods,
- mechanisms for evaluating the recommendations - whether rule based, modelled with math, data-mined, pure AI-based or hybrid.

Recommender systems can recommend different entities: content, actions or whole sets of actions, the strategies.

Different recommendation systems work in different environments and they should be adapted to their domain constraints, data availability and context.

There are different sources of data that recommendation system can process to perform recommendation. Whether it’s internet travelsal, telemetry, biometry, database querying, etc. Some of them require data preprocessing: denoising, sampling, reducing dimensionality, etc.

There are also various methods of processing the recommendations. The systems can use classification and clusterization algorithms. They can mine the data in search for associative and action rules or use expert entered rules. Ranking recommendations can be done by applying matemathical models or artificial intelligence algorithms, etc.

2.1. Classification by subject of recommendation

When comparing recommendation systems it is important to know what do they actually recommend. Different systems offer recommendations on different level of system usage abstraction. In this section I explain

---

1 Harré, “Thinking with Models”
2 Note that this is my own subjective choice of classification measures, compare it with ones in Ricci et al., Recommender Systems Handbook, pp. 187-216.
3 Strategy recommendation was inspired by Tzacheva and Ras, Association Action Rules and Action Paths Triggered by Meta-actions.
2.1. Classification by subject of recommendation

Figure 2.1: Diagram presenting hierarchy of recommendation systems by subject of recommendation (content, action or strategy).

how these systems can recommend content, actions and strategies, I also present an examples.

2.1.1. Content recommendation systems

Content-based systems are the most popular ones, their logic is based on correlating different items of content together. The most common functions of content-based systems are: providing personalized content search, providing contextualized recommendations of items, and providing personalized recommendations of items.

These recommendation systems are useful for supporting navigation in content-heavy systems, like: e-commerce platforms, knowledge bases, encyclopedies, social networking sites, item rating sites, like Netflix or Filmas-

Most of the time, content-based systems do not require real-time data analysis and correlation. Often these actions can be performed off-line, and often include:

1. correlating items together,
2. correlating users together
3. correlating users with items together, traversing graph of dependencies between users and items for recommendations.

2.1. Classification by subject of recommendation

Classifying a system as content-based system doesn’t imply the mechanisms for correlating items and users together.

Figure 2.2: BBC World News - correlation of articles by their subject. Example of content to content correlation.

Figure 2.3: Youtube.com - movie (content) recommendation. Example of explaining the recommendation choice.
2.1. Classification by subject of recommendation

Figure 2.4: Amazon.com - book recommendation. Example of frequent pattern content correlation.

Figures 2.2, 2.3 and 2.4 show examples of recommendations based on content.

2.1.2. Action recommender systems

Recommendation systems, as defined in chapter [1.1] are not limited to content-based ones. As a auxiliary system the can recommend not only content-items but interactions too.

Computer systems’ user interface can be defined as a mean to provide user with ability to use and act upon system’s functionalities. System should provide user with actions, which user is able and authorized to perform, and which lie within system’s functionality.

The recommendation system’s role can be broaden to:

— evaluating usefulness of the actions to user,
— presenting most useful actions to the user.

Action recommendation systems can be found in places when real-time user action support is required, but they are not limited to this use case. These systems, are found useful in:

— application’s features guides,
— feature usage tutorials,
— computer games,
— decision support systems,
— medical treatment recommender systems

There are various mechanisms that these systems can process recommendations with. Often there is no data for application to process, or recommendation needs to be performed quickly. Action recommendation often use expert-provided math models [7] or adaptive artificial intelligence to make decision.

Content-based systems can be defined as a subpart of the action-based systems, where only limited set of content-centric actions are supported (view, read, etc.).

[8] Tanaka et al., “GA-based decision support system for multicriteria optimization”
2.1. Classification by subject of recommendation

Figure 2.5: Wikipedia contextual request for citations. Based on item profile (Encyclopedic article has no citation, etc.).

2.1.3. Strategy recommendation systems

The strategy recommender systems offer higher level of abstraction than action-based ones. These systems are supposed to understand and support user choosing the strategy – set of actions, or use-cases that will help users succeed at their current goal.

This type of systems provide users with recommendation which is a set of actions. Recommended actions can be sequential or parallel, leading to goal, which are useful for system’s user. Strategy recommender systems need to:

— understand results of performing multiple actions,
— understand user’s interests and goals,

Strategy recommender systems would be useful in majority of the computer systems, but they are expensive in development and computation. Sophistication of those systems results in heavy-use of manually defined expert rules. Most of contemporary strategy-based systems provide user with simple hard-coded wizard-like functionality.

Two examples of such recommendation systems, could be:

— medical treatment recommendation,
— computer game adaptive AI.

The medical treatment recommendation can be based on meta-action-rules discovery. User’s purpose is to normalize patient’s health condition, and minimize costs of treatment. Action supported by the system are, inter alia: performing a test and administering drug or treatment,

Simple action-rule discovery is not enough. Recommendation of single drugs and treatments may be misleading and cause deterioration of patient’s health. Single test won’t help normalizing patient’s health status. The proper approach is to embrace knowledge about interactions between different tests, drugs and treatments, and propose a meta-action, a strategy that would normalize patient’s health condition.

9 Bergman et al., “Personal Wizards: collaborative end-user programming”.
2.2. Classification by environmental constraints on recommendation

Other example of recommendation mechanism that makes predictions, to enhance gameplay experience is Adaptive Dramatic Pacing\textsuperscript{11} for AI game-director - a technique present in “Left 4 Dead” computer game. It’s purpose is to create dramatic game pacing in a cooperative zombie killing FPS game. It modulates game mechanisms using informations about user feelings. The actions that are available for the game logic, are:

- Introducing more intense game-play, by putting more zombies to attack the players.
- Wait for players to regenerate - creating suspension before next zombie wave.

The AI director monitors intensity of gameplay, including: player avatar’s health status, time since last zombies were killed, etc. Then it tells game to spawn more zombies, or create suspension “Relax” time.

2.2. Classification by environmental constraints

Recommendation systems work in different environment, which impose different constraints onto the system. Here is a list of some of those constraints:

- environmental constraints,
- data acquisition constraints,

\textsuperscript{11} Booth, \textit{The AI Systems of Left 4 Dead}
Figure 2.7: Diagram presents: top (Desired Population) – how many enemies would AI game-director put into gameplay. middle (Survivor Intensity) – a measure of player’s feelings. bottom (Actual Population) – amount of enemies AI game-director actually put into gameplay, after considering player’s feelings.

— temporal constraints,
— cardinality of domain constraints.

Recommendation system can work in different environments: deterministic and nondeterministic, continuous or discreet, stable or dynamic environment.

Different environments pose different data acquisition problems: If environment is dynamic, will the recommendations be valid in the future? Is acquired data big enough to provide relevant recommendations? How to analyze massive amounts of data?

When dealing with dynamic environment, there is often a need for frequent or fast recommendations. How to process data fast enough? How to leverage time series as a stream of information?

Structure of system, that recommendations are made in, is equally important. It is easier to provide recommendation coverage when there is limited number of possible actions or content items. There are different methods to deal with big or infinite amount of possible recommendations – different for continuous and discreet domain.

The most important constraint, that drives development of recommendation system, is the supported system, and it’s environment. Systems, supported by recommendation system, can require different amount of user interaction. There are systems where user’s reaction speed is important, and recommender systems should embrace that. If system acquires lots of data, then the computational complexity, the amount of resources needed to recommend, can pose a real problem.
There are lot of systems that can leverage recommendation mechanisms. Some of them are:

- computer games,
- knowledge bases,
- social websites,
- e-commerce portals,
- SCADA (Supervisory Control And Data Acquisition) industrial systems,
- stock trading support systems, etc.

All these systems require different methods of data gathering and analysis. While knowledge bases can leverage deferred processing of textual data, the computer games require fast, real-time decision support with hard time constraints.

## 2.3. Classification by prediction knowledge source

I came out, with another way of recommendation systems classification: Classification by prediction knowledge source. This approach focuses on

---

12 Huang, *Intelligent Decision Support: Handbook of Applications and Advances of the Rough Sets Theory*
the implementation of recommender systems and lets understand differences among any Artificial Intelligence (AI) systems.

Recommendation Systems (RS) logic can be implemented in different ways, as the definition of RS is made from a black-box perspective. Recommendation systems are part of AI, but still, what affects a system’s logic structure the most is:

— the amount of expert knowledge entered to the system by people,
— the amount of knowledge discovered using computer learning and analysis,

Following subchapter describes different sources of knowledge in recommender systems: expert knowledge, computer learning based and hybrid solutions.

Figure 2.9: Diagram presents example systems and mechanisms that they are based on, in the space of expert knowledge and computer learning usage. Colored areas represent different types of systems: green – trivial systems, yellow – systems based on mainly on expert knowledge, purple – systems based mainly on computer learning, blue – hybrid systems.

2.3.1. Recommendation systems based on expert knowledge

Artificial intelligence is still not widely used in the computer industry. It is hard to spot intelligent algorithms in programs, that we use everyday (text-editors, etc.). There are lot of reasons for this:
2.3. Classification by prediction knowledge source

— Some systems must provide extreme levels of safety, and their work
must be robust and predictable.\(^\text{13}\)

— Human brain is better at strategic thinking\(^\text{14}\) than computer artificial intelligence\(^\text{15}\)\(^\text{16}\).

— People know the domain of the problem that is solved by system.

— It is extremely hard for computer to understand abstract concepts.

Given that, in most cases there is a need for expert to provide enough
amount of knowledge, that will enable the system to solve the problem. I
present here, some of the ways that experts can enter the knowledge into a
system.

2.3.2. Expert knowledge based recommendation - Rules and
Ontologies

The simplest mechanism of providing intelligence to a system is to pro-
vide it with manually entered expert’s knowledge\(^\text{18}\). One of approaches for
entering expert knowledge into a computer system is:

— to provide system developers with specification and hard-code
knowledge into system’s code-base,

— to provide expert with Domain Specific Language (DSL), to provide
description of the knowledge so it can be interpreted by system.

— to provide expert with special user interface for entering the knowl-
dge.

— to provide expert with ontology to describe generalized structure
and relations of problem domain. etc.

In means of recommendation systems, this mechanisms enable systems
to provide sophisticated recommendations to the user. In systems with big
amount of data, this mechanism is too expensive, due to it’s manual nature.

The rules inside the system can be represented as a simple modus
ponens implications or more complex ontological relations. Experts provide
system with implications and rules, which can be translated into contents’
correlation or directly into recommendations for the user.

Examples: "If the user X buys stock of company Y, recommend user with
buying stock of company Y.” or "If user is a first-time user, recommend him
filling in personal info.”

This type of implications let experts provide precise knowledge into a
system. The shortcoming of that is not providing means to enter more
generalized knowledge or notion of uncertainty. There should be always
some doubt, about heuristic model that expert is providing, as it may be

\(^{13}\) Kruse, Borgelt, and Runkler, "Robust Learning in Safety-Related Domains".
\(^{14}\) note: strategic thinking, not strategic analysis
\(^{15}\) Henden, "Intuition and its role in strategic thinking".
\(^{16}\) Mintzberg, "The fall and rise of strategic planning".
\(^{17}\) Searle, "Is the Brain’s Mind a Computer?".
\(^{18}\) Clancey, "The epistemology of a rule-based expert system—a framework for expla-
nation".
based on not representative set of facts. Therefore, providing system with full knowledge can be impossible or very expensive.

### 2.3.3. Expert knowledge based recommendation - Mathematical model-based and heuristic-based

Another way of providing recommendation system with a knowledge is implementing mathematical model that is prepared by experts. Expert provided models can be based either on scientific model or expert's heuristics. Mathematical model is also more formal way of performing recommendation, it enables: working with unlimited(or great) number of possible actions, strict way of ranking recommendations, and making formal proof of the system's operation.

Working with unlimited number of possible actions is often connected with making a choice of solution in some mathematical space. This kind of problems often are often found in optimization or automatic control.

There are problems, which solution do not require dealing with continuous, unlimited choices. Examples of problems, which solution is found in limited sized, discrete space, are:

- Choosing what article to read, from 100 articles available on newspaper’s website.
- Choosing courses on the university, that student will appreciate the most,
- Choosing one of four roles in a cooperative computer game, that would most likely match player’s interest and game style.

Here are some discrete and continuous space problems, that have huge or virtually unlimited number of possible solutions: choosing optimal cyclic route between cities or choosing optimal resource allocation.

Although mathematical model-based approach offers strict control over the recommendations, it has flaws. The biggest problem with this approach is the problem of entering knowledge by an expert. The expert is required to not only have expertise in problem’s domain, but also he must be able to: describe the problem quantitatively and model the problem using complex mathematics.

Often, description of the model requires expert to have mathematical skills. Fortunately many problems are already modeled with mathematics. The ones, that are not, pose a problem for experts. One of the solutions for making mathematical models easily enterable is usage of domain-specific languages and fuzzy logic.

---

19 Mitra and Greenberg, *Mathematical models for decision support*.
21 Tanaka et al., “GA-based decision support system for multicriteria optimization”.
22 like traveling salesman problem or Google Maps route recommendation.
23 like knapsack problem.
Figure 2.10: *Choice of glass sheet cuts optimized by PerfectFit® software. The software can model this problem as a knapsack problem.*

Domain specific languages (DSL) provide experts with ability to describe model using domain-specific naming. The need to know complex mathematics and programming languages.

Fuzzy logic is a logical system that lets experts enter rules for mathematical models without the need to operate on mathematical equations\(^2^4\). With fuzzy logic: Entering strict values and thresholds is turned into entering linguistic variables - ex. *"high temperature"*, instead of “*T*\(>130\text{F}^\circ\)*”. It’s possible to enter easily readable rules instead of mathematical formulas and equations - ex. *"if the market is optimistic, then recommend producing more"*.

Thanks to fuzzy logic and it’s ability to create domain-specific naming, it is easy to create domain-specific language (DSL) for experts to enter knowledge\(^2^5\). This is especially effective in industries and sectors, where non-technical people are required to use computer.

\(^{24}\) Zadeh. *Fuzzy logic= computing with words*.
\(^{25}\) Fowler. *Domain-specific languages*.
2.3. Classification by prediction knowledge source

2.3.4. Recommendation systems based on computer learning and data analysis

The expert-knowledge based system are growing in complexity and become more expensive to create. Artificial Intelligence (AI) systems are reasonably easy to maintain and cheap. The popularity of data-analysis and Big Data systems\[26\] shows that the systems using more sophisticated data-driven computer algorithms are the future of computing.

The recommender systems can use data analysis and computer learning for extracting the knowledge out of the data sets available to the system. Some of them are: system’s content, system’s environmental data, user data, and user behavior tracking data.

![Diagram of data flow through knowledge system.](image)

The computers can replace experts on different levels of knowledge system depth. Computers can perform\[27\]

- data acquisition. Experts can analyze the and create proper rules.
- data acquisition and analysis. Experts can use analyzed data patterns and information to create rules.
- data acquisition, analysis and adaptation. Where rules entered by experts are adaptively optimized by the AI.
- pure AI-based recommendation. Recommendation without the need of expert to enter knowledge.

Computer systems perform better at data acquisition than people do. The data can be acquired in various ways, including: querying databases, reading sensors, through telemetry, through biometry, through internet traversal, etc.

The data then must be converted and stored into a computer-friendly digital format, like: tables, time series, matrices, etc.

Computers are used for statistics, data analysis and business intelligence for years now. They can analyze data much faster than people do. There are many ways that data can be analyzed, here are some examples: association and frequent pattern rule mining, sequential rules and patterns mining, action rules extraction, clustering, etc.

There should be enough data and analysis for experts or AI to make a decision regarding best recommendation.

\[26\] Lohr, "The age of big data"
\[27\] Fayyad et al., "Advances in knowledge discovery and data mining"
2.3. Classification by prediction knowledge source

2.3.5. Adaptive recommendation models

There are situations, where purely analytical model is not available, experts come with suboptimal solutions and rules-of-thumb to solve some problem. Adaptive models can use experts’ domain knowledge as a starting point. Using AI and optimization they can adapt to the system’s environment, giving better solution.

One of examples of adaptive model are neuro-fuzzy systems. Neuro-fuzzy systems use neural networks to perform changes on linguistic variable and rules, so the system can perform better.

![Diagram of adaptive recommendation system](image)

Figure 2.12: Example data flow in adaptive recommendation system. The expert entered data acts as a entry point of further system optimization.

---

28 Jang, Sun, and Mizutani, “Neuro-fuzzy and soft computing—a computational approach to learning and machine intelligence [Book Review]”

29 Liu and Lampinen, “A fuzzy adaptive differential evolution algorithm”
2.3.6. Artificial intelligence based recommendation models

With properly designed architecture, it is possible to abstract recommendations, actions and user profile in a way that they could be fed into self-learning artificial intelligence system. Using proper user experience metrics it is possible to create feedback loop for the AI to verify and assess it's knowledge.

Figure 2.13: Example flow of AI based recommendation system.
3. Recommendation mechanisms

Usage of computer learning, data analysis and correlation is very popular approach to recommendation in content-based systems. In this section of the book I will explain why data correlation mechanisms can be:

— user-wise,
— content-wise, (content-based)
— social-wise, (collaborative filtering-based, neighbourhood search-based)
— mathematical and explorative.

This chapter abstract away from the implementation of those mechanisms. It describes ideas on how to correlate recommendations with the user.

The user-wise mechanisms focus on the processing of user profile data. The content-wise algorithms try to correlate system contents/actions together. The social-wise mechanisms analyze social neighbourhood of the user, in order to suggest him with recommendations. I also describe how all this approaches can be abstracted by applying mathematical model of recommendation.

3.1. User-wise mechanisms

The term user-wise mechanisms, refers to mechanisms that uses user profile to give recommendations

1 Golemati et al., “Creating an ontology for the user profile: Method and applications”
3.2. Content-wise mechanisms

— and many more, like:
  — geolocation, etc.
— User behaviour tracking data\(^2\):
  — The system’s content that user like’s the most,
  — content that user shared with others,
  — content that user frequently uses,
  — content that user ‘liked’, etc.
  — track of action’s that user performs in the system,
  — web pages visited,
  — buttons clicked, etc.
  — dynamics of user’s actions, etc.

Figure 3.1: Trivial example of user-wise recommendation. System knows (black arrow) that Jacek plays League of Legends frequently, so the system recommends (green arrow) him with playing more.

Given this type of data a user profile can be prepared, and digitized. This user profile can be used to give user personalized recommendations, that:
match his personal interests, geolocation, civil status, etc.

This mechanism poses problem for the recommendation system, because it needs certain amount of data (the “critical-mass” data) to perform meaningful personalized recommendations for the user.

3.2. Content-wise mechanisms

Content-wise mechanisms are recommendation mechanisms that use system content correlation to prepare recommendations for the user\(^3\). This type of mechanisms require analysis of relations between different system content, like:

\(^2\) Ahmad Wasfi, “Collecting user access patterns for building user profiles and collaborative filtering”.

\(^3\) Ricci et al., Recommender Systems Handbook, pp. 73-100.
3.2. Content-wise mechanisms

Figure 3.2: Google Gmail Ads personalized recommendation based on email correspondence content. Example of user profile (email-content) to content (ads) recommendation.

— newspaper articles,
— wikipedia pages,
— movies,
— images,
— user comments,
— books, etc.

This mechanism usually serves for advising the user about the content he should use later. The recommended content is often directly connected with the content that user already used or visited. This mechanism doesn’t require to know user profile. It is most convenient for gathering “critical-mass”\(^4\) data about the user, without prior knowing him. The “critical-mass” data is the minimal amount of data to perform personalized recommendations for the user.

In order to compute correlations between content items, system needs to extract features out of the items. This can include:

— text-mining - inc. creating document-term matrix, etc.
— image processing - inc. edge detection, segmentation, computing invariant moments of different segments, etc.
— meta-data processing - inc. usage of ontologies, text meta data, links inside hypertext documents, etc.
— heuristics, etc.

The point of computing features is later analysis of system’s content items. Basic later analysis should answer following questions: Is this content item relevant? What other documents is this item correlated with?

\(^4\) Shardanand and Maes, “Social information filtering: algorithms for automating ‘word of mouth’”
Figure 3.3: *Example of content-wise book recommendation*. *System doesn’t know the user*. Because the user looks (dotted line) at *Clean Code* book, the system can recommend book’s sequel.

The answers to the questions should be stored in computer-friendly format, so the knowledge can be easily processed by the recommendation system.
3.3. Social-wise mechanisms

Collaborative filtering is another way of filtering informations. It leverages social interactions between system users. Many contemporary computer systems have some social features. Whether it is: social network, software-as-a-service application or enterprise software, there is a notion of social interaction present. It can be:

— relations in social networks, like: Facebook or LinkedIn,
— being a member of workgroup in collaboration software, like Trello or Basecamp,
— having similar occupation to other employees of the same company.
— using some publicly available service, etc.

Figure 3.4: Example of social-wise recommendation. Jacek has two friends (stripped lines): Sly and Poirot, Sly knows Mr. Miyagi, and Poirot likes (dotted line) to play Domino. System analyzes this situations and comes up with two recommendation.

---

The point of social-wise mechanisms is to leverage the data about multiple users to perform recommendations. Social-wise mechanisms are similar to user-wise mechanisms, but they use more information.

The social-wise mechanism can find similar users, to the current user of the system, and use their profiles to perform recommendations. The method of similar user search can vary.

These mechanisms can correlate users to actions by using data from many user profiles. Correlation can be done using anonymous measures, using only content profiles, or using user profiles, correlating users together. Also, social-wise mechanisms leverage the social relations between users, e.g. recommending content items or actions that user’s neighbours use or perform.

### 3.4. Social-graph and neighbourhood-based mechanisms

Mixed approach leverages the most of the mechanisms described before. There is a large number of different connections types and correlations between people that exist in modern systems. The social graph is one of the best ways to describe connections between people, content-items, and actions. Social graph is a special type of graph, where:

**Vertices** are people and content-items, like: movies, articles, recipes, etc.

**Edges** are actions performed on content-items by actors of the system.

The idea behind the social graph makes it possible to create indirect relations through network of people. Because user is embedded in the social network, traversing the graph provides information about content and actions that can be relevant to the user.

The mechanisms that use traversal of similar or connected items are called neighbourhood-based algorithms. Social-graph recommendation is one of the examples of neighbourhood-based algorithms.

### 3.5. Mathematical approach

Mathematical models can provide us with a strict way of ranking recommendations, as recommendation can be represented as a relation:

\[ \Gamma_U : A \rightarrow R \]  

Where:

---

6 Wasserman and Faust, *Social network analysis: Methods and applications*.

3.5. Mathematical approach

Figure 3.5: Example graph traversal (red arrows), resulting in indirect recommendation via social graph. Yellow rectangles describe relations.

\( U \): is the user,
\( \Gamma_U \): is recommendation for user \( U \),
\( A \): is set of user actions or system’s content to be recommended,
\( R \): is set of real numbers that represents a usability of the action.

This relation can be implemented as any function, which can take various parameters, including user profile features, system features, environmental features, etc. if they can be measured or computed.

Given such a function, system will have to evaluate recommendation, searching for optimum in some multi-dimensional space. For this purpose computers can use any optimization algorithm ranging from analytical optimization to heuristics.
4. Crowdsourced Knowledge Base for Fast Information Search

This chapter of describes a problem of the thesis. This includes, describing:

— the problem, requirements and constraints that it is posing,
— typical solutions of the problem, that does not use recommender mechanisms,
— my solution to the problem, that is knowledge base with recommendation system,

I present classification of the problem and solution, using measures that I presented in the first part of the book. I explain my approach and decisions made, while analysing the problem.

4.1. Problem description

The problem, which solution I present in this thesis, is the problem of creating accessible crowdsourced knowledge base for fast information search. Here’s breakdown of problem’s name:

**Information search** - is accessing relevant knowledge about some problem or domain.

**Fast information search** - means to provide users with really fast access to the knowledge.

**Accessible information search** - is providing users with simple accessible interface, that can be used by people with broad range of computer-usage capabilities.

**Knowledge base** - is the solution, the source of knowledge that users can access.

**Crowdsourced knowledge base** - a knowledge base, that lets every user to manage and enter informations[1].

Table [4.1] provides brief information about problem, expected solution’s functionality, users, environment, constraints, requirements, and possible applications.

---

[1] vide. Wikipedia, etc.
4.2. MediaWiki system as a solution

4.2. MediaWiki system as a solution

Table 4.1: Table with brief information about the problem.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Crowdsourced Knowledge Base</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution functionality</td>
<td>Providing users with knowledge, Providing means for experts to enter knowledge.</td>
</tr>
<tr>
<td>Users</td>
<td>users - broad range of users, with different level of computer usage skills, experts - users entering the knowledge.</td>
</tr>
<tr>
<td>Environment</td>
<td>Crowdsourced data, anonymous, anxious, one-time users.</td>
</tr>
<tr>
<td>Usage constraints and requirements</td>
<td>Lots of data, divided into categories, lot of users, minor data changes couple times a day, required high availability for the users, required fast information filtering and search, no initial data to identify users.</td>
</tr>
<tr>
<td>Example problems</td>
<td>&quot;Safe internet use&quot; knowledge base, Illnesses, symptoms and treatments knowledge base Knowledge base about different food diets, products and recipes.</td>
</tr>
</tbody>
</table>

4.2. MediaWiki system as a solution

History of solving problem presented in this thesis (knowledge base) is long. Solutions that were present in the past vary from academic libraries to manuals available online as PDF files and content-management systems. I was challenged with taking the solution bit farther, introducing computer interface to knowledge base and providing convenient crowdsourcing and recommendation.

Typical solution of the problem for crowdsourced knowledge base is MediaWiki system, which is the base for Wikipedia - online encyclopedia. The MediaWiki system main properties are:

**Crowdsourcing** - making it possible for the users to manage the content of the encyclopedia.

**Flexible as a Content Management System (CMS)** - it provides extensible user interface for updating the data, it operates on different types of media.

**Advanced user management and monitoring**.

**Web scalability** - Wikipedia systems operate around the world, serving thousands of requests every second for millions of users.¹

In next subsections I describe these properties and functionalities. I write about flaws and advantages of using MediaWiki system.

¹ Miller, *A Look Inside Wikipedia’s Infrastructure*.
4.2. MediaWiki system as a solution

4.2.1. Crowdsourcing and user management in MediaWiki

MediaWiki system was created to be a base of Wikipedia, which is big web service with encyclopedic knowledge. By the design Wikipedia is very open, it accepts anyone to update the encyclopedia and fill in the missing data. The updates in the system:

**can be updated by registered wikipedians** - who are experts interested in some domain. They created an account, and are responsible for some content categories, etc. By writing articles they gain reputation inside the wikipedians community.

**can be updated by anonymous users** - anyone who enters MediaWiki system can propose updates to the content. It is properly marked, and will be due to review by wikipedians.

**are monitored by administrators and moderators** - they check the content for illegal material, technological abuse, flooding, hacking, etc.

A whole hierarchy of crowdsourcing roles is available for managing system contents. Access to the system can be managed and secured by password, anonymous usage can be disabled, etc.

4.2.2. MediaWiki as Content Management System

MediaWiki serves as a text content management system for millions of users around the world. The content is organized into a pages (wiki pages), which are articles written by contributors. As a CMS system MediaWiki provides:

- content editor,
- media support,
- navigation,
- authorization mechanisms,
- administration engine, etc.

Content editor provides special MediaText markup, special convenient language for formatting the articles. It provides $\LaTeX$ markup for mathematical equations. There are also WYSIWYG plugins available for the MediaWiki.

MediaWiki provides functionality of hosting ana managing many media types, including: images, videos, sounds, presentations, PDF files, etc.

There are hyperlinks throughout wiki pages providing navigation. They provide catalog-like navigation. There are also categorisation features, that let manually group wiki pages togather.

MediaWiki authorization mechanism, allows to identify content author. The content can be locked from public exposure, etc. There is also fullFEATURED administration engine for managing: users, content, logs, user groups, etc.

---

4.2. MediaWiki system as a solution

4.2.3. MediaWiki Scalability and Technology

MediaWiki system provides decent scalability. Serving Wikipedia, it responds to 50 thousands request per second\textsuperscript{4}. MediaWiki system is written in PHP. It supports working on multiple servers in parallel. It supports various caching techniques, including: client side caching, page-caching\textsuperscript{5}, data object caching\textsuperscript{6}, etc. It can use MySQL, PostgreSQL or Ingres for databases. It is based on Apache Core for serving HTTP requests. It’s modular architecture is highly extensible.

MediaWiki system serves wiki pages as a HTML web pages. Each of HTML pages, images and other media is easily cacheable. Use of modern technology makes MediaWiki very scalable system.

4.2.4. Flaws of MediaWiki as a solution

MediaWiki system appears to be great choice for knowledge base, but it doesn’t necessarily fulfil all the requirements of the problem posed in this thesis. There are problems involving MediaWiki system, that make it unusable for fast knowledge access. Here are those problems:

— It is system for encyclopedic knowledge.
— It has no contextual search. Search mechanisms work poorly.
— It provides no personalized search.
— It has dated user interface design, not suitable for the solution.

Firstly Wikipedia and other wiki systems are designed primarily to provide encyclopedic knowledge. When user is browsing through an encyclopedia, he usually knows what he’s looking for. The Solution should provide user with all the knowledge he needs. While user doesn’t know what he is looking for, MediaWiki systems will not be suitable for him.

Secondly the text search features in MediaWiki systems are unusable. Wikipedia users are forced to use universal search engines\textsuperscript{7} in order to quickly find any relevant information. MediaWiki's search is not contextualized and it doesn't personalize results of the search. MediaWiki doesn’t have any recommendation mechanism inside, and there is no article correlation mechanism, other than hyperlinks inside articles.

The search engine provides only strict string matching algorithm (any typo or mistake results in zero search results). It also provides no querying language (despite lot of meta-data in the articles, there is no ontology or querying language for the users to work with).

Finally, Wikipedia has dated Web 1.0 design\textsuperscript{8}. Modern computer users are more demanding, when it comes to design. Although there are plugins that let the administrators change the look & feel of MediaWiki system, it still has many flaws. The design is not interactive or responsive. It

\textsuperscript{4} Miller, \textit{A Look Inside Wikipedia’s Infrastructure}.
\textsuperscript{5} like Squid servers
\textsuperscript{6} like Memcached
\textsuperscript{7} like Google Search or Bing
\textsuperscript{8} compare with O’reilly, \textit{What is web 2.0}
4.3. Development approach decision

When analyzing any problem, one should think about the users and environment that the solution will work with and the resources for solving the problem. There’s always two ways of solving problems: (1) building solution from the scratch, or (2) using existing technology and building on top of it.

4.3.1. Building solution from scratch

The first choice, building from scratch, has its pros and cons. When we program our solution, we are much more confident with the quality of the software, we know exactly how it works. The solution’s software architecture can be precisely tailored for the problem’s domain. Knowing all the constraints and requirements, we can easily develop a system with proper scalability.

There are also disadvantages of this approach. Firstly, the system has many functionalities, that need to be developed, so this approach requires more work. Secondly, developed system will not be field-tested on production environment, so solution will probably require more maintenance at first.

4.3.2. Building on top of existing computer systems

The second choice – building on top of existing technology, is much faster approach. One can take different systems, and by gluing them together, create a viable solution for the problem. Different parts of the system can cover different functional requirements.

Although the development of such solution can be very fast, there are disadvantages of this approach. Firstly, there is a certain amount of knowledge, that is required to use legacy code of different developers. Secondly, while the cost of developing functionalities is dramatically lower, there is a need to integrate different systems, again, dealing with legacy code. Lastly, maintenance and hosting of many parallelly working system can significantly increase the amount of computer resources needed by the system to work efficiently in production environment.
### 4.3. Development approach decision

<table>
<thead>
<tr>
<th>Developing system from scratch</th>
<th>Development on top of existing solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Development</strong></td>
<td></td>
</tr>
<tr>
<td>- Huge amount of functionalities to develop.</td>
<td>+ Small amount of code, needed only for integration of different systems.</td>
</tr>
<tr>
<td>+ Trust in code quality.</td>
<td>- High cost of learning different systems.</td>
</tr>
<tr>
<td></td>
<td>- High cost of dealing with legacy code and system architectures,</td>
</tr>
<tr>
<td><strong>Maintenance</strong></td>
<td></td>
</tr>
<tr>
<td>+ Full comprehension of developed code, so maintenance is easier,</td>
<td>- Many different maintenance mechanisms, that need to be serviced.</td>
</tr>
<tr>
<td>- Needed to develop own maintenance mechanisms</td>
<td></td>
</tr>
<tr>
<td><strong>Resources</strong></td>
<td></td>
</tr>
<tr>
<td>Scalability can be easily ensured, by choosing proper technologies</td>
<td>- Many systems, with no consistent background technologies (different databases, libraries, etc.) require more resources</td>
</tr>
</tbody>
</table>

Table 4.2: Table of advantages and disadvantages of two different development approaches.

### 4.3.3. Making development choice

Even basic analysis of requirements of system to be created, tells that this would be a big project if I chosen to develop it from the scratch. After consideration I came to this conclusions:

- Writing crowdsourced content management (CMS) system is unnecessary work.
- MediaWiki is open-source and provides full CMS functionality.
- MediaWiki provides crowdsourcing mechanisms.
- Available recommender system solutions (read: content-based or collaborative-filtering information filtering systems) are too expensive to integrate, too complicated, too excessive in functionality to consider them as a part of the solution.
- MediaWiki is viable, robust choice for a base of the system,

Writing crowdsourced CMS system is unnecessary work, as this thesis subject is recommender systems, not CMS systems, authorization mechanisms or markup languages. Adapting existing system would let me focus on implementation and experiment with recommendation mechanisms.

MediaWiki provides all the required CMS functionality, being flexible web application. It has the crowdsourcing functionality and provides markup language that seems to be perfect for further extension.

One of the advantages of MediaWiki system is its scalability, it works both on massive systems, like Wikipedia, and smaller systems, like small
4.4. Personalized information filtering with fast profile building

When developing knowledge base’s recommender system, one must think of its functionalities and constraints. In order to enable users to quickly acquire information, the system uses recommendation system. The recommendation systems needs some user profile to present personalized recommendations for the user. It needs to know answers to following questions: Who is the user? What informations are most relevant to this user?

4.4.1. Recommender system challenges

Acquiring user profile is hard, knowing that most of the users are one-time users. This implies no authorization when viewing content - because it would create unnecessary barrier for first-time users, etc.

There is limited possibility of personalized system usage tracking, when users do not login and they use the system occasionally. One cannot track their behavior or create proper user dossier. There is not enough personal data.

4.4.2. Fast profile building

Providing users with relevant information requires users to provide minimal amount of information about who they are. This can be done by asking users about their place within problem’s domain. Example applications and required information:

**Safe internet usage knowledge base** - user should provide information about websites and applications that he is using, in order to know what are the threats that await him and how can he defend himself against them.

**Medical knowledge base** - user should provide information about symptoms that the ill person has, in order to know what are the probable diseases that he has and to acquire tips on what to do.

---

9 Ex. when user searches the knowledge base for information about computer viruses, he should enter informations about his computer: operation system version, etc. This would enable limiting information about viruses to those that attack such operation system, etc.
Recipes recommender system - in order to get personalized recommendation about recipes, user should provide information about: diets that he is on (ex. vegan, low fat, etc.), products that he has in the fridge.

Data in a form of selection of domain objects (ex. recipes, diets, products, diseases, drugs, symptoms, etc.) which are relevant to the user is enough to perform basic recommendation. There are also additional informations that can be leveraged by the system. Web browser request data is an example which provides:

- IP - which enables basic localisation of the user,
- browser agent info - which gives type of user’s web browser and his operating system, etc.

Figure 4.1: Diagram presenting data flow in the solution. User enters some basic information about his place within problem’s domain. The system recommends relevant content to the user.
### Table 4.3: Table presenting requirements for solution and how my system fulfills them.

<table>
<thead>
<tr>
<th><strong>Problem requirements</strong></th>
<th><strong>How the system fulfills the requirements</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Features</strong></td>
<td>User builds his profile, and is recommended with wiki articles with knowledge in them.</td>
</tr>
<tr>
<td>Providing users with knowledge,</td>
<td>Providing means for experts to enter knowledge.</td>
</tr>
<tr>
<td>Providing means for experts to enter knowledge.</td>
<td>MediaWiki content management system CMS takes</td>
</tr>
<tr>
<td><strong>Users</strong></td>
<td>Simple knowledge system interface with no authorization (for building profile and for information search).</td>
</tr>
<tr>
<td>Users - broad range of users, with different level of computer usage skills.</td>
<td>MediaWiki crowdsourcing features let wikipedist enter the knowledge. Extended MediaWiki markup for entering knowledge</td>
</tr>
<tr>
<td>Experts - users entering the knowledge.</td>
<td></td>
</tr>
<tr>
<td><strong>Knowledge system</strong></td>
<td>Fast user profile building for one-time users.</td>
</tr>
<tr>
<td>Crowdsourced data, anonymous, anxious, one-time users,</td>
<td></td>
</tr>
<tr>
<td><strong>CMS features</strong></td>
<td>MediaWiki for CMS scalability and articles frontend.</td>
</tr>
<tr>
<td>Lots of data, divided into categories, lot of users, minor data changes couple times a day, required high availability for the users,</td>
<td></td>
</tr>
<tr>
<td><strong>Recommender system features</strong></td>
<td>Special light recommendation interface for building user profile, and getting recommendations</td>
</tr>
<tr>
<td>Required fast information filtering and search, no initial data to identify users</td>
<td></td>
</tr>
</tbody>
</table>
4.4. Personalized information filtering with fast profile building

<table>
<thead>
<tr>
<th>Classification type</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical classification</td>
<td>Hybrid content and collaborative-filtering based</td>
</tr>
<tr>
<td>By subject of recommendation</td>
<td>Content recommendation</td>
</tr>
<tr>
<td>By environment and constraints</td>
<td>Stable www environment with deferred processing of knowledge.</td>
</tr>
<tr>
<td>By knowledge source</td>
<td>Hybrid with bias towards expert-knowledge based. There are both computer data correlation and manual expert-knowledge entry.</td>
</tr>
<tr>
<td>Mechanisms used</td>
<td>Expert rules, Content-wise mechanisms and Social-wise mechanisms</td>
</tr>
</tbody>
</table>

Table 4.4: Table presenting classification of the knowledge base recommender system, using measures described in the first part of the book.
4.4. Personalized information filtering with fast profile building

Two tables 4.3 and 4.4 describe the solution. The first one, describes how different requirements are implemented inside the solution. The second one describes the classification of recommender system, that will be used within the knowledge base (the solution).
5. Implementation

The functional problem analysis, that I presented in previous chapter of the book, was performed using a black-box approach. This was helpful for analysing what features should product have.

This part of the book focuses on the white-box approach, where system will be analyzed and described including some implementational details and architecture of the solution. I describe architecture of the system, certain functionalities of the system, development decisions and effects of working on the recommendation system.

5.1. Data flows

Important element of the system’s implementation is the information about flow of the data through system components. This subsection explains, where the data comes from, where it is processed and where it is stored afterwards.

My recommendation system’s data flow consists of three main flows, which are:

— Wiki CMS\(^1\) data flow,
— knowledge mining data flow, and
— knowledge system querying data flow

Wiki data flow is the process that is performed by wikipedians and contributors, providing the crowdsourced content to the end users. Knowledge mining data flow is the process of turning information about the content into relevant recommendations for the users. Knowledge system querying data flow is about serving recommendations to the end user.

There are many ways to present communication inside computer system, I prefer the data-flow diagrams. The diagrams consist of two kinds of nodes and edges. Circle nodes represent data entry or data processing nodes. Box nodes represent data stores.

5.1.1. Wiki data flow

The system’s content is entered thorough wiki editing interface. Both anonymous users and registered wikipedians can enter the content into the system.

\(^1\) content management system
5.1. Data flows

5.1.2. Knowledge mining data flow

Knowledge mining is a process, where wiki article correlations are computed. In this flow, following data is processed: textual content of wikipedia articles, expert knowledge embedded in special tags on wiki pages, user tracking data.

These processing, let the system prepare:

— content-wise recommendation - by correlating articles using text-mining,
5.1. Data flows

— social-wise recommendation - by converting frequent patterns, in usage-tracking data, into correlations between articles.
— experts’ knowledge-based recommendation - by mapping of experts’ knowledge into the correlations between articles.
— search engine - creation of term-document matrix for search index.
— preparing data cache - by preparing data structures easily queried by the recommender system.

This flow is one of the most important features, of the system. I describe every step of processing knowledge in following chapters.

5.1.3. Knowledge query interface data flow

Users must be able to query the knowledge system for the recommendations. Querying is made by performing a request to knowledge system API\(^2\). The request consists of: user profile and query string.

The user profile is set of documents and categories(domain objects) relevant to the users, that were selected by him. They are passed to as an array of identifiers. Query string helps to additionally filter the results, for relevant information.

As a response to a request, the user should get a recommendation and the request should be tracked for further data-mining.

![Diagram](image)

Figure 5.3: *Diagram presenting the flow of the data when receiving recommendation request from the user.*

The process is presented on figure 5.3. Receiving the request, the knowledge system API, validates the request, fetches proper data from knowledge

\(^2\) Application Programming Interface

\(^3\) Knowledge API is a component that serves as a backend for recommendation. It responds to recommendation requests form user browsers, etc. It’s implemented as JSON API.
5.1. Data flows

system cache (prepared by knowledge discovery), saves the request for further analysis, ranks the documents, and returns relevant headlines to the user.
5.2. Architecture

This section is an eagle-eye view on the system. In following subsections present:

— set of components that I used inside the system,
— approaches to integrating those systems,
— explanation of different architectural choices.

Getting to know the architecture, will let you understand the functionalities of the components, that I describe in next subsections.

This section explains dependencies between different components of the recommendation system. It also explains some architectural choices. Following paragraphs and subsections explain elements of the diagram from top to the bottom.

5.2.1. System architecture overview

A diagram below presents system’s architecture from eagle-view perspective.

Figure 5.4: Diagram presenting overview of the systems architecture. Users (at the top), interfaces(yellow), backends(blue) and datastores(dark blue)
5.2. Architecture

5.2.2. Users and interfaces

The system serves three kinds of users, which are: system’s administrator, wikipedian - creator of the content, and the end user - who uses the system for education.

Each of them have their own interfaces that they interact with. There are six user interfaces (UI) that face the user, these are:

**End user’s recommendation system UI** - which enables the user to enter his user profile, and efficiently browse through the results of his queries.

**End user’s MediaWiki UI** - which serves as a content display for the system. This is where the user reads the articles.

**End user’s MediaWiki recommendation plugin** - which extends functionality of MediaWiki UI with additional recommendation capabilities.

**Wikipedian’s MediaWiki user interface** - which enables the crowdsourcing of system’s content. It consists of article editor and media management tools.

**Administrator’s MediaWiki interface** - an interface for administering user privileges, monitoring the system’s resources, fighting abuse, etc.

**Administrator’s data processing interface** - interface for scheduling data-mining jobs, cache updates/invalidations, etc.

These interfaces are crucial for providing required functionality of the system.

5.2.3. Backends

Background knowledge-discovery jobs, MediaWiki’s API and knowledge system’s API are the backend of the system. These are the components, that are crucial parts of data flow processing.

Knowledge API responds to end user’s requests for recommending articles. It is robust algorithm, that fetches information from the recommendation cache, and serves it to the users.

MediaWiki’s API serve wiki content, and enable wiki knowledge extraction, through REST interface. It provides all the CMS functionalities that are required.

Background jobs - are set of algorithms that perform data processing. There is whole subchapter explaining these jobs.

5.2.4. Data stores

The system’s data is stored in two separate databases: MySQL and MongoDB. This increases system’s setup time and amount administration that system requires. This approach also has advantages.

MySQL ships with MediaWiki. It provides relational data storage, for wikipedia content and user management. This database can be easily swapped with the PostgreSQL system.
5.2. Architecture

The MongoDB is a document database, it is a storage of JSON-formatted scheme-less data inside named data sets, called collections. MongoDB as a NoSQL database is much faster than relational databases, but doesn’t support full ACID (atomic, consistent, isolated and durable) transactions. The MongoDB is used for: storing a knowledge cache for knowledge API, storing user tracking data, storing temporary data of data-mining algorithms.

The reason for using MongoDB is fact that, it is easy to install, manage and scale. MongoDB runs on many platforms and doesn’t require lot of resources to operate.

5.2.5. Integration

The system’s components are written with four different programming languages. It implies the need of integration using methods shared for each of the programming platforms. These are: database interfaces, and ability to perform HTTP requests.

Given that MediaWiki system was developed, before this project came into conception phase, the development of other system components is constrained by the way they interact with MediaWiki system.

There are four different planes of integration between the components, these are:

- MediaWiki’s REST API
- MediaWiki’s HTML rendering component
- MediaWiki’s database
- Knowledge processing data flows’ integrating data sets, in MongoDB
- Recommender system’s knowledge cache

MediaWiki’s REST API enables HTTP wiki data fetching, for data processing jobs. MediaWiki’s HTML rendering component integrates with JavaScript plugin, enabling recommender features inside wiki articles. MediaWiki’s MySQL database is one of the sources of data for data processing jobs. MongoDB holds all the temporary data that is used by data processing jobs. It also serves as a store for recommender knowledge cache.

5.2.6. System configuration

System requires four types of nodes, these are:

- static content delivery node,
- web nodes,
- database nodes,
- worker nodes.

First type of nodes serves static content to the end users. These nodes can be placed in a content delivery network for best performance. They don’t store any valuable or private information.

Web nodes are the run:
5.2. Architecture

— MediaWiki,
— knowledge API,
— load balancers,
— reverse proxies and accelerators, etc.

MediaWiki runs on PHP stack, knowledge API runs Ruby server written with Sinatra framework, load balancers are optimizing request flow to other web nodes. Reverse proxies and accelerators can cache HTTP requests, etc.

Database nodes run MySQL and MongoDB, they require bigger hard disks and computation power. Their configuration can leverage master-slave write policies, sharding and replication. Memcached object cache servers can also be used for caching database queries and computation results.

Worker nodes run the knowledge discovery scripts. In order to perform computation faster, they can be assigned with bigger resources (memory, CPU, etc.).

In test environment only one computer serves as all these nodes. In the grown stage production environment, when service is heavily used, all these nodes and functions can be separated. But they should be scaled step-by-step as the system grows.

Figure 5.5: Scalable configuration for grown stage production environment. The lines show direct integration between the components.
5.3. Knowledge Processing

This subsection describes methods of implementing recommendation mechanisms. System features two kinds of recommendation mechanisms: content and social-based ones. In following paragraphs I describe how the recommendation-knowledge cache looks like, and what are the mechanisms for filling it up with relevant knowledge.

The backbone of recommendation mechanisms are:

— textual article correlation,
— expert knowledge rules, and
— frequent pattern correlation.

Following sub chapter explains how are they implemented in the system. I also try to explain the line of inference, that lies behind each of the mechanisms.

5.3.1. Knowledge cache system

Knowledge cache stores quickly accessible information for Knowledge API. The information is prepared by the data processing jobs. The cache consists of:

— term-document matrix cache,
— document-document correlation cache.

The term-document matrix cache is the base of search engine features of knowledge API. In order to fetch all the documents correlated with given set of terms, the software has to perform one database query. This is important, as MongoDB where information is stored, is document database and no data joining is available.

The records in term-document matrix cache consist of JSON records in following format:

Listing 5.1: Listing of term-document matrix cache record JSON format

```json
{
    _id: ... , // unique identifier of record
    t: ... ,   // the term
    p: ... ,   // the wiki page id
    nam: ... , // title of the wiki page
    cat: ... , // category of the wiki page
    cor: ...   // correlation of wiki page to the term
}
```

This format requires lot of memory to be stored, because there are lot of cache records. It also ensures fast querying, due to it’s denormalized form.

The document-document correlation cache provides data for the content-based recommendation. Each record of the cache provides digitized information about correlation of two wiki pages. The records in the document-document corelation cache have following JSON format:
5.3. Knowledge Processing

Listing 5.2: Listing of document-document cache record JSON format

```json
{
    _id: ... , // unique identifier of record
    a: ... ,  // the page id
    p: ... ,  // id of the page that "a" page is correlated with
    nam: ... , // the p page title
    cat: ... , // category of the "p" wiki page
    cor: ...  // correlation of "a" and "p" page
}
```

Knowing the formats of knowledge in the cache, let's us know what is expected outcome of the knowledge processing jobs. How are the caches filled, is the question answered in next paragraphs.

5.3.2. Text-mining document correlation and creating search index

The first mechanism, that I describe is the textual article correlation. The inference behind this mechanism is following:

**Similar articles** - Articles X and Y are textually similar, user selected document X, we should recommend document Y to the user, because of similarity.

Data flow, described in following paragraphs, is responsible for matching similar documents together, and creating search index for full-text search. The flow consists of following stages:

1. Extracting the data from databases and MediaWiki API,
2. Extracting features from the data,
3. Saving term-document matrix, as a search index.
4. Selecting proper subset of features for document correlation,
5. Correlating articles using selected features,
6. Saving the knowledge into knowledge cache.

MediaWiki holds the data in MySQL database. Articles are stored as wiki pages. The algorithms must extract following data: wiki page title, wiki page identifier, wiki page categories. and article text contents.

The title, identifier and textual content is easy to fetch from the database. Wiki page category information, is occluded inside a data table, and is not easily accessible. This is why the solution uses MediaWiki API call, to extract the categories.

Given all the required data is extracted, the algorithm saves the results in MongoDB NoSQL document database. Which serves as temporary data store, throughout all the data processing jobs.

In order to correlate articles using traditional methods, they must be described with a set of labeled numbers - the features. The smaller the number of significant features, the faster the correlation algorithms will work.
5.3. Knowledge Processing

There are different possible features that can be extracted out of a text. Occurrence number of terms, words, and letters, the length of the document, etc., can be examples.

There are norms that work on single document, the local norms, and global norms that use data about all the documents in a system. I tested couple of norms, and settled with the simplest one. I use tf norm - term frequency norm, which is described with equation:

\[ tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \]  \hspace{1cm} (5.1)

Where \( n_{i,j} \) is the number of occurrence, of term \( (t_i) \) in a document \( d_i \), and denominator is sum of occurrences of all the terms in the document. Before \( n_{i,j} \) can be computed, all the terms inside a document, must be stemmed. Stemming let’s different variation and conjugations of same word be counted as a single term, and summed together.

Every document processed has many different words, which effects in very large set of terms, that describe each document. This number can be reduced by filtering stop-words, and removing sparse terms. Another difficulty is that different document have different set of words inside them. In order to fix those issues it is useful, to consider document-term matrix.
5.3. Knowledge Processing

Table 5.1: Example document-term matrix. Column marked green, represents a stop-word column. Column marked red, is a sparse term column.

<table>
<thead>
<tr>
<th>Article</th>
<th>$t_1$</th>
<th>$t_2$ - &quot;the&quot;</th>
<th>$\ldots$</th>
<th>$t_k$ - &quot;cardiovascular&quot;</th>
<th>$t_{k+1}$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Article $d_1$</td>
<td>0.0</td>
<td>0.9</td>
<td>0.3</td>
<td>0.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Article $d_2$</td>
<td>0.11</td>
<td>0.9</td>
<td>0.42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>Article $d_p$</td>
<td>0.12</td>
<td>0.9</td>
<td>0</td>
<td>0.11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Article $d_{p+1}$</td>
<td>0.03</td>
<td>0.9</td>
<td>0</td>
<td>0.23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>

Document-term matrix rows describe different documents (here: articles), and the columns describe terms. Each matrix cell $(i, j)$, is the value of the norm for term $t_j$ inside document $d_i$.

In order to find significant set of features describing all the documents, one must remove too sparse and too dense columns from the matrix. The sparse columns describe terms, that are rare, and do not have value when comparing documents\(^4\). The dense columns contain mostly stopwords - words common in one’s language (like "the", "a", "on", "when", etc.) that are also not valuable for comparing documents\(^5\).

Term-document matrix is transposed document-term matrix. It is used to correlate terms with documents. Given term-document matrix as a data structure, we can easily access every document, that has terms of our choice. So term-document matrix is a good starting point for full-text search index.

System creates term-document matrix along with selecting the features. It saves it into a search index cache, to be accessed by the knowledge API.

Document-term matrix is used for correlating similar documents. The correlation is a measure of documents similarity. Correlating of $N$ wiki articles, consists of checking dependencies between $N^2$ different documents. There are different ways of comparing textual documents. The one that I use is the cosine similarity\(^6\).

---

\(^4\) rarely any document has these terms.

\(^5\) almost every document has them, these are ubiquitous terms.

\(^6\) Tan, Steinbach, and Kumar, *Introduction to Data Mining* pp. 500.
5.3. Knowledge Processing

\[
\text{similarity} = \cos(\phi) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \tag{5.2}
\]

Where:

- \(A\): is the vector of attributes(features) the of document A,
- \(B\): is the vector of attributes of the document B,
- \(A_i\): is the value of attribute i of the document A,
- \(B_i\): is the value of attribute i of the document B,
- \(n\): is the number of attributes that documents, are described with.
- \(\phi\): if the vector of attributes of document A and B, would be considered a vectors in some euclidean hyperspace. The \(\phi\) would be a angle between those vectors.

As presented in this equation [5.2], similarity is a norm that can be understood as measure of similarity between two sets of attributes. The results of similarity computation are stored in MongoDB for further analysis. The whole process can be optimized by using clustering algorithms.

5.3.3. Expert knowledge parsing

The textual data correlation is just one of the mechanisms that provide knowledge for recommending articles. The second one is expert knowledge. Any wikipedian, that will choose to do so, can manually enforce positive or negative correlation, between any two wiki pages. This enforced relation is not necessarily symmetric. The inference behind this mechanism is following:

**Expert knowledge** - User chosen article X and article Y is referenced by an expert inside article X, we should recommend(or discourage) user to choose document Y.

**Text-search** - User searches for terms T, there is set of documents, that have term T inside them - A, we should recommend documents inside A to the user.

MediaWiki is full-stack content management system. The easiest way to introduce expert-knowledge entry is to embed this mechanism in existing MediaWiki content-management mechanisms. This is why I choose entering special text-tags inside a wiki page, as a knowledge entry mechanism.

**Listing 5.3: Expert-tag format for MediaWiki CMS**

```html
<!−− Expert | Correlated page title | 1.2 −−>
```

The wikipedians while writing articles, are accustomed with special MediaWiki Markup language that lets, them format the documents, and introduce different meta-data. My idea was to implement my own markup element that will let experts interact with recommendation mechanism. This
5.3. Knowledge Processing

markup element is a special tag, that is inserted inside given wiki page. Listing 5.3 presents the format of the expert-tag. Where:

<expert | is beginning clause,
Correlated page title - is the title of correlated page,
| - is separator,
1.2 - is the value of the enforced correlation,
-> - is the ending clause.

The expert-tags provide simple way to correlate two articles by hand. They are case insensitive, and they can have positive or negative values.

The expert-tag processing job, looks like this:
1. The data is fetched from MediaWiki system. This step is shared with textual data correlation.
2. Articles’ text contents are searched for expert tags.
3. The correlations are inserted into knowledge cache automatically.

Figure 5.7: Diagram visualising sequence of operations used to extract expert-tags and save the correlation data to knowledge cache.

5.3.4. Frequent patterns document and terms correlation

The last mechanism of recommendation that I covered in this thesis is the frequent pattern social-wise recommendation.
Users builds the profile. His browser sends recommendation requests to the knowledge API. These requests enclose the user profile. They consist of:

- Articles that user selected- as wiki page identifiers,
- Category of documents that he’s looking for - as string, and
- Search bar query string - which helps further contextualizing the content.

The system builds a big database of these historical requests. It can use these requests to perform social-wise recommendations to the users. The recommendation inference can work in following way:

**Frequent patterns** - There are a lot of people, who selected set of documents, with document X and Y inside. Because the user selected document X, we should recommend document Y to the user.

**Popularity** - There are a lot of people, who selected document X, we should recommend document X to the user.

In order to perform such a recommendations, the system needs to analyze the tracking data. Also, we must keep in mind, that: The data can get old. What was popular one year ago, can be irrelevant today, and that, popularity recommendation, must work even when no user profile is present.

![Diagram presenting how data flows through frequent pattern processing job.](image)
5.4. Recommender system's interface - Knowledge API

Given the amount of tracking data can be big, there is a need to minimize computational and memory complexity of the frequent pattern recommendation algorithm. In order to do so, the algorithm, looks like this:

1. For each tracking record $R$ that is $t$ seconds old, do:
   a) $A = Ids_t + id_x$ - where $Ids_t$ is set of wiki page identifiers from record $R$, and $id_x$ is an empty identifier.
   b) For each pair $(id_1, id_2)$ inside $A$, do:
      i. correlate positively document identified $id_1$ and document identified by using relevance evaluation function $f(t)$.

The problem of dropping over time relevance of tracking data is resolved, by introducing relevance evaluation function $f(t)$, where $f: \mathbb{R}_+ \rightarrow (0, 1)$. The $f$ function should be a smooth function that diminish the relevance of correlation, between documents inside a tracking record. Correlation should be smaller, as the tracking record is older. Example implementation of this function can be $f(t) = 2^{-kt}$, where $0 < k < 1$, ex. if $k = 1/3600$ it would make correlation of half strength if the tracking record was one hour old.

When there is no user profile available the system still returns valid results. It is because there is always a hidden wiki page identifier $id_x$, which points to no page. This enables the system to correlate documents to $id_x$, based on popularity, even when the knowledge api query is empty. $id_x$ is added to every query automatically.

5.4. Recommender system’s interface - Knowledge API

Knowledge API is a backend of recommender user interfaces. It provides user interfaces with proper recommendations. API request consists of:

— optional list of documents selected by the user - part of user profile,
— optional category that user is interested in - part of user profile,
— optional search query string - part of the full-text search engine.

Listing 5.4: Example Knowledge API request URL.

http://localhost:4567/query/?categories=Fixes&doc_ids=19,16,20,32&query_string=LinkedIn

Example request is shown on listing 5.4. It provides category “Fixes”, list of wiki page id’s 19, 16, 20 and 32, and a query string "LinkedIn".

Given user sends a request the API:

1. parses query parameters from URL,
2. validates query,
3. performs query,
4. saves the query for further frequent pattern analysis,
5. renders JSON response to the request.

Query parameter parsing extracts array of categories, array of article identifiers and a query string.
Query validation is used for abuse and flood control. Validation may include checking number of documents in the user profile, checking if given categories are legal, checking length of query string, etc.

Performing a query is connected with:
- fetching data from knowledge cache,
- ranking retrieved recommendations (documents), and
- filtering and merging results.

![Flow of data when performing a query.](image)

**Figure 5.9:** Flow of data when performing a query.

<table>
<thead>
<tr>
<th>Correlation type</th>
<th>Ranking Reward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Textual correlation</td>
<td>+Textual cosine similarity</td>
</tr>
<tr>
<td>Expert knowledge</td>
<td>+ Expert provided value</td>
</tr>
<tr>
<td>Frequent pattern correlation</td>
<td>+ Time-adjusted tracking data relevance function value</td>
</tr>
<tr>
<td>Category correlation</td>
<td>+1 if wiki page category matches the queried one</td>
</tr>
</tbody>
</table>

**Figure 5.10:** Table of weights for document ranking algorithm.

Figure 5.9 presents the flow of the data when performing a query. Three kinds of parameters (user profile’s documents, user profile’s categories and query string) are used to fetch documents from the knowledge cache. They can be ranked using different methods: summing correlations using weights shown in table 5.10. Documents are filtered for duplicates, which are removed, and a response is provided to the user.
5.5. User interface

As described in section 5.2.2, there are 6 user interfaces that whole system uses. These are:

```
Listing 5.5: Example JSON response from Knowledge API. Notice the correlation values inside results objects.

```{  
    "query":{
        "categories":[
            "Fixes"
        ],
        "doc_ids":[
            19,
            16,
            20,
            32
        ],
        "query_string":"LinkedIn"
    },
    "result":[
        {
            "id":5,
            "title":"LinkedIn",
            "categories":[
                "Apps"
            ],
            "correlation":5.6500702093071091
        },
        {
            "id":2,
            "title":"Goldenline",
            "categories":[
                "Apps"
            ],
            "correlation":2.163807504828603
        },
        {
            "id":3,
            "title":"Email",
            "categories":[
                "Apps"
            ],
            "correlation":1.4472936291653964
        }
    ]
}
```
5.5. **User interface**

— end user’s recommendation system UI
— end user’s Mediawiki UI,
— end user’s MediaWiki recommendation plugin - which extends functionality of Mediawiki UI with additional recommendation capabilities.
— wikipedian’s MediaWiki user interface - which enables the crowdsourcing of system’s content. It consists of article editor and media management tools.
— administrator’s MediaWiki interface - an interface for administering user privileges, monitoring the system’s resources, fighting abuse, etc.
— administrator’s data processing interface - interface for scheduling data-mining jobs, cache updates/invalidations, etc

In this subsection I briefly describe their functions and present example screenshots of their look and feel.

### 5.5.1. MediaWiki user interface

![Figure 5.11: MediaWiki wiki page’s look and feel.](image)
End user’s MediaWiki UI is the standard MediaWiki user interface that lets users read the wiki pages. Articles within MediaWiki UI are presented as formatted wiki pages, with navigations and media.

Wikipedian’s user interface is a typical interface for content management in MediaWiki. It provides simple MediaWiki markup editor, discussions and comments features for wikipedians, etc.

5.5.2. Recommendation system’s UI

![Example screenshot of recommendation system’s UI](image)

End user’s recommendation system UI is an interface that lets user build the user profile. It lets him choose: categories and domain objects, that are relevant to him, or ones that describe his position the most.
5.5. User interface

5.5.3. MediaWiki recommendation plugin

![Figure 5.13: Screenshot of contextual MediaWiki recommendation plugin. When selecting a text (here: 'unauthorized access'), system automatically searches for articles connected with given words.]

Typical MediaWiki UI is extended with special JavaScript plugin, that lets user get contextualized recommendations, when reading the articles.

When user is browsing the content recommendations are available, when: he’ll click the "Further reading" button on the side of the screen, and when he’ll select a text. The first recommendation is done using article correlation recommendation, and the other one is done with full-text search engine.

Both Recommendation system’s UI and the plugin use JavaScript heavily. JavaScript and HTML5 enabled creating this responsive and dynamic interface. UI uses Ajax and JSON to communicate with Knowledge API.

5.5.4. Data processing interface

The data processing jobs are controlled via console. The jobs are implemented as Rake\(^7\) tasks. These task can be run directly via console, or they can be scheduled with cron-like programs to schedule the work. All the jobs’ descriptions can be found using ‘rake -T’ command.

\(^7\) ‘make’ dependency-system’s alternative for Ruby programming language.
5.6. Possible development

System should organically grow with the data and use cases, that current users, wikipedians and administrators will came up with. The growth of this system can be considered in two areas: scaling and features.

Scaling is a problem of: serving content to more users, and serving more contents. Architecture that I proposed looks to be easily scalable vertically. The knowledge processing seems to be bigger problem, as in current stage of development correlation algorithms are not optimized for time complexity. The amount of correlations to be made and stored in database, for \( N \) wiki pages, is \( N^2 \). But it can be optimized by using clustering, etc.

Moving the knowledge processing, from manually written scripts to knowledge processing systems, like Apache Mahout, etc. would make the processing part of the system much more scalable. Writing the jobs by myself, was good idea for completely understanding the mechanisms implemented in the recommender system.

Secondly, there is an area for growing number of features. This can include using more informations to build user profile, using different mechanisms to recommend knowledge to the users(social-graph, etc.).

The system was designed as a proof-of-concept of knowledge base with accessible interface. It can be developed further by anyone, as code is available publicly on the internet.
6. Summary

The purpose of this thesis was to create an recommendation system for crowdsourced knowledge base. In order to do so, I accustomed readers with different types of recommendation systems. I presented the problem of creating knowledge base, that can provide user with relevant informations. Finally, I shown an implementation of such knowledge base.

In the first chapter I describe what are the recommender systems – systems or set of features, which main purpose is to support human-computer interaction, by giving advices.

In second and third chapter I present ways to classify recommendation systems and mechanisms that can be used inside them. This is a merithorical background for the second part of the book, supported with a literature study. There are also my own contributions to recommender systems classification.

Fourth chapter describes a problem: Creating recommendation mechanisms for knowledge base with fast profile building, expert and computer-extracted knowledge. Computer programs need to understand the user more, in order to be accessible for wider audience. My recommendation system helps users with exploring the data, giving personalized advices about articles to read, using minimal amount of information about the user.

The fifth chapter describes the implementation of my recommendation system. System’s data flows are explained. Reader can find there: a brief overview of system architecture, description of knowledge-mining methods and interfaces used between various components of the system.

Recommender systems are trending topic inside IT industry. Number of functionalities of computer systems grow. Users need to operate in more and more complicated environments. Software businesses want to enable broader audience to be able to use their software. There is a need for supporting the users. Recommendation systems are one of the solutions to this problem. There is a lot of possibilities of expanding recommender systems. I hope this thesis helped you understand and harness these opportunities.
Bibliography


[14] Jyh-Shing Roger Jang, Chuen-Tsai Sun, and Eiji Mizutani. “Neuro-fuzzy and soft computing—a computational approach to


List of Figures

2.1 Diagram presenting hierarchy of recommendation systems by subject of recommendation (content, action or strategy)................................. 5
2.2 BBC World News - correlation of articles by their subject. Example of content to content correlation................................................................. 6
2.3 Youtube.com - movie (content) recommendation. Example of explaining the recommendation choice................................................................. 6
2.4 Amazon.com - book recommendation. Example of frequent pattern content correlation................................................................. 7
2.5 Wikipedia contextual request for citations. Based on item profile (Encyclopedic article has no citation, etc.).................................................. 8
2.6 Microsoft Word paperclip. It provides users with annoying contextualized help................................................................. 9
2.7 Diagram presents: top (Desired Population) – how many enemies would AI game-director put into gameplay. middle (Survivor Intensity) – a measure of player feelings. bottom (Actual Population) – amount of enemies AI game-director actually put into gameplay. after considering player’s feelings................................. 10
2.8 Diagram presenting different environmental factors that can affect recommender systems................................................................. 11
2.9 Diagram presents example systems and mechanisms that they are based on, in the space of expert knowledge and computer learning usage. Colored areas represent different types of systems: green – trivial systems, yellow – systems based on mainly on expert knowledge, purple – systems based mainly on computer learning, blue – hybrid systems................................................................. 12
2.10 Choice of glass sheet cuts optimized by PerfectFit© software. The software can model this problem as a knapsack problem................................. 15
2.11 Diagram of data flow through knowledge system................................................................. 16
2.12 Example data flow in adaptive recommendation system. The expert entered data acts as a entry point of further system optimization................................................................. 17
2.13 Example flow of AI based recommendation system................................................................. 18
3.1 Trivial example of user-wise recommendation. System knows (black arrow) that Jacek plays League of Legends frequently, so the system recommends (green arrow) him with playing more................................................................. 20
3.2 Google Gmail Ads personalized recommendation based on email correspondence content. Example of user profile (email-content) to content (ads) recommendation................................................................. 21
3.3 Example of content-wise book recommendation. System doesn’t know the user. Because the user looks (dotted line) at Clean Code book, the system can recommend book’s sequel. ........................................ 22

3.4 Example of social-wise recommendation. Jacek has two friends (stripped lines): Sly and Poirot. Sly knows Mr. Miyagi, and Poirot likes (dotted-line) to play Domino. System analyzes this situations and comes up with two recommendation. ........................................ 23

3.5 Example graph traversal (red arrows), resulting in indirect recommendation via social graph. Yellow rectangles describe relations. ........................................ 25

4.1 Diagram presenting data flow in the solution. User enters some basic information about his place within problem’s domain. The system recommends relevant content to the user. .............................. 33

5.1 Flow of the article management performed by wikipedians and anonymous users. ........................................ 38

5.2 Flow of the knowledge discovery. It serves for content recommendation. ........................................ 38

5.3 Diagram presenting the flow of the data when receiving recommendation request from the user. ........................................ 39

5.4 Diagram presenting overview of the systems architecture. Users (at the top), interfaces (yellow), backends (blue) and datastores (dark blue) ......... 41

5.5 Scalable configuration for grown stage production environment. The lines show direct integration between the components. ........................................ 44

5.6 Diagram visualising sequence of operations used to extract textual data from articles, stored in MediaWiki ........................................ 47

5.7 Diagram visualising sequence of operations used to extract expert-tags and save the correlation data to knowledge cache. ................................. 50

5.8 Diagram presenting how data flows through frequent pattern processing job. ........................................ 51

5.9 Flow of data when performing a query. ........................................ 53

5.10 Table of weights for document ranking algorithm. ........................................ 53

5.11 MediaWiki wiki page’s look and feel. ........................................ 55

5.12 Example screenshot of recommendation system’s UI. This example system lets user get informations about how to safely use and browse World Wide Web. The boxes highlighted green are the ones chosen by user. The red ones are recommended for the user. ........................................ 56

5.13 Screenshot of contextual MediaWiki recommendation plugin. When selecting a text (here: ‘unauthorized access’), system automatically searches for articles connected with given words. ........................................ 57
List of Tables

4.1 Table with brief information about the problem. .......................... 27
4.2 Table of advantages and disadvantages of two different development approaches. ...................................................... 31
4.3 Table presenting requirements for solution and how my system fulfills them. .............................................................. 34
4.4 Table presenting classification of the knowledge base recommender system, using measures described in the first part of the book. ......... 35

5.1 Example document-term matrix. Column marked green, represents a stop-word column. Column marked red, is a sparse term column. ....... 48