A Recommender System for a Legal Search Engine

J.W. Nelen
J.M. Nederlof
T.R.D. van Graft
M.P.C. van der Werf

Delft University of Technology

TU Delft

blue tick

A Recommender System for a Legal Search Engine

Bachelor Thesis at Bluetick

by

J.W. Nelen
J.M. Nederlof
T.R.D. van Graft
M.P.C. van der Werf

To obtain the Bachelor degree of Computer Science and Engineering
at the Delft University of Technology,
Faculty of Electrical Engineering, Mathematics and Computer Science

To be defended publicly on Thursday January 28, 2021

Project duration	November 9, 2020 - January 29, 2021	
Thesis committee	C.C.S. Liem	TU Delft, Coach
	T.A.R. Overklift	TU Delft, Bachelor Project Coordinator
	K. Kooijman	Bluetick, Client

This thesis is confidential and cannot be made public until January 29, 2021

Preface

This report concludes the TI3860 Bachelor-project course, a mandatory course in the bachelor program of Computer Science and Engineering at the Delft University of Technology.

We want to thank everyone involved that made our project at Bluetick possible. In specific, we would like to thank Cynthia, our coach from the university, who was a great source of inspiration during the whole project. Her constructive feedback highly contributed to the quality of the final product. Additionally, we would like to thank Kasper, the CPO of Bluetick, for his enthusiasm, faith, availability and patience during this project. Lastly, we would like to thank everyone else at Bluetick namely Koen, Martijn, Thijs, Daniel, Lotte, Pelle and Marlies for their frequent feedback and insights crucial for this project's success.

J.W. Nelen
J.M. Nederlof
T.R.D. van Graft
M.P.C. van der Werf
Delft, January 2021

Executive Summary

Bluetick offers a juridical research platform that enables lawyers to search for cases and jurisprudence efficiently. Most Dutch legal alternatives are still old-fashioned search engines. Bluetick wants to move towards a zero-search-based approach where the system learns about the user's preference and provides them with recommendations. For the user, this means that they have to spend less time searching for cases while still finding all the relevant material. To reach this goal of zero-search, the quality of the recommendations must be high. Therefore improvements in this area are believed to result in a more lucrative product.

This report describes the process of improving the version of the recommender system that was already implemented by Bluetick. The main contributions are evaluated by their effect on the recommender system, and their role in creating a more maintainable, extensible and transparent product.

The first contribution of the team was a refactor of the old system. Using classes and interfaces, the new version makes it easier to do advanced computations on the results, while the interface makes it easier for Bluetick to add additional parts on which recommendations can be based. Secondly, similar to many existing webshops, the new system provides the user with insight into why items are recommended. Lastly, the user is now able to provide the system with relevant law articles at the start, so that the recommender system can give recommendations before the first search.

Contents

1 Introduction

The goal of this project is to create an improved recommender system for Bluetick's search engine. First, in this introduction, additional context about Bluetick, their market and their product will be given. The last section gives an overview of the rest of the report.

1.1. The Client

In late 2019, Bluetick was created by a recently graduated master student group. They met each other on a master that combines data science and entrepreneurship. One of the courses of that master was the assignment to set up a new startup. The founders of Bluetick wanted to go all in, and they started Bluetick.

1.2. Bluetick's market

The founders decided to disrupt the legal search engine market. Professionals in law, e.g. lawyers, will spend a lot of their time looking for cases similar to the one they are currently working on. The documents of these cases are called jurisprudence. The user reads these old cases to gain more insight into how judges interpret specific laws, and the cases also provide different points of view for a lawyer. Currently, the search engines that exist for this purpose are not very user friendly and often limited to keyword searches.

1.3. Bluetick's product

Bluetick's search engine allows for search by keywords, European Case Law Identifier (ECLI), and larger parts of a text. Next to this search engine, they have a recommender system. The system creates a score for other cases based on the user's search history, relevant marked cases, and pieces of text that the user highlighted inside cases. All this information is stored in a collection. A user can have multiple collections, most likely one for each of his clients or for a certain law area. Based on these combined scores, an ordered list is made with recommendations which is shown to the user.

Bluetick aims to have a search-less system, where the user only does a minimal amount of searches. To reach this goal, they want to use a recommender system that gives the user cases based on his collection. If the recommendations are accurate enough, the user will have to perform less manual searches, because the recommender system automatically presents relevant cases that can be used for the current collections.

Bluetick's system uses a front end in React, which is a JavaScript framework for reusable components that enables building fast web application. The backend is written in Django, a REST framework for Python.

1.4. How would a customer use Bluetick's search engine?

A typical customer is a law firm that will pay a monthly subscription to Bluetick to make use of the search engine. For every case that an employee of a law firm, e.g. a lawyer, is working on, he can

create a new collection. The lawyer then enters his first search, to find the first relevant cases and to give the recommender system information. The recommender system will suggest new similar cases to the lawyer, that he will use to build up his collection. The lawyer will highlight applicable parts of the text, and he will mark cases as 'relevant' when he finds one.

The goal of this is that the lawyer can build up an understanding of the law and its interpretation. An example of this would be to look up similar cases to the one the lawyer is working on, so he can give his client an indication of the final verdict. Another example is that the lawyer can use the approach that other lawyers took in a similar case.

1.5. How do other legal search engines work?

Existing Dutch legal search engines are still old-fashioned. They can be used for searching with an ECLI or keywords. The legal search engines are not smart. For example, when searching for murder with a baseball bat, the search engines will not return murder cases with a hammer, although the cases would be very similar.

Internationally, search engines are a lot of smarter. For example, in the United States, Artificial Intelligence is already being used to make search engines better at finding similar cases. On first sight, it is a weird idea: Why are these techniques that are used in the United States not yet used in The Netherlands? Firstly, this split is caused by the difference in lawsuits. A case in the United States works differently than in The Netherlands. Secondly, text-based search and Natural Language Programming (NLP) approaches are optimized for the language they were designed for, which means that companies that have smart search engines can not easily add Dutch law and cases.

1.6. Structure of the report

In the next chapter, chapter 2, this report gives the original plan that the team created together before the project started and how the plan changed during the project. In chapter 3, the original research report is given, where the team describes the research they have done prior to implementation. The next chapter, 4 gives a more concrete problem definition, taking the research into account. Chapter 5, 6 and 7 explain what the team has implemented and contributed to Bluetick's product. After this, in chapter 8, the development process of the project is explained and evaluated. In chapter 9 the team gives an outlook for Bluetick, with recommendations for the future. Finally, in chapter 10, a conclusion is given about the whole project.

2

Project Plan

This chapter entails a compact plan with criteria for the project, a timeframe to implement it and some information regarding the process, meetings and more. This plan was created at the start of the project and served as a guideline and beacon for the project. The team discussed this plan with both the TU Delft coach and Bluetick to check and clarify the requirements and expectations. The original project plan is included in Appendix C. What follows is a summary of the original project plan and the main points in which the team deviated from it.

2.1. Time frame

In this project, the new features are an addition to an existing system; therefore, the time before a feature reaches production will be relatively short. This is also reflected in the planning, as the implementation will start from the first week.

2.2. Acceptance criteria

The original acceptance criteria were:

1. There is a working version of a recommender system
2. There is an improvement in the quality (which will be based on the metrics) of the recommendations compared to the already existing baseline
3. The recommendations provided by the recommender system are verifiable and reproducible
4. The code is sufficiently maintainable by developers of Bluetick
5. The code and documentation is clean and easy to read
6. There is a research report that motivates the different design choices clearly and concise and could be used as an additional explanation of the product.

These criteria helped the team prioritise feature requests from both the TU Delft and Bluetick. Furthermore, these criteria eventually lead to the 3P plan that is discussed in Chapter 4, which allowed the team to structurally plan each feature.

2.3. Deviations

After starting to work on the project, the team relatively early discovered a limitation resulting in some of the acceptance criteria to receive more focus from the team than others. To be more specific, the team decided that it would be more valuable for the project to lay a foundation for easy improvements to the recommendations' quality instead of building upon the existing one.

The application of more advanced ensemble methods such as a machine learning model was considered the most logical way to improve. However, the system was considered not yet ready for

those methods. The refactor performed in this project would have been a pre-requisite to make it reasonably maintainable and extensible. Secondly, the accuracy of these models would depend on having a sufficiently large training and test set. The team did not expect that this would be viable within the given time frame and, therefore, prioritised other areas. In Chapter 4 the new problem definition is explained in more detail.

3

Research

As this project's goal was to improve the current recommender system, the team decided to do extensive research on recommender systems. The first section explains recommender systems and when they are used. In section 3.2 the different types of recommender systems are described. After this, some more context is given about the use of implicit feedback in recommender systems in section 3.3. The current situation concerning data collection at Bluetick is discussed in section 3.4. The results are then presented in 3.5 and this chapter finishes with the research conclusion in section 3.6

3.1. Recommender system fundamentals

This chapter gives a short overview of the well-known recommender systems. Furthermore, it will delve into the reason why recommender systems are necessary. The last section will briefly sum up a few problems that often arise when creating a recommender system.

3.1.1. Why is a recommender system necessary

Since the rise of the age of big data are recommender systems, an effective solution to the problem of information overload. Recommender systems are also a solution to information overload in the field of law, where more and more lawsuits are documented online, as mentioned in [1]. Lawyers want to find the one lawsuit that helps their clients to win a lawsuit. This search creates the need for an efficient searching algorithm; however, to search someone would need a suitable amount of domain knowledge or have luck with guessing the right keywords. A recommender system helps to solve those needs; it can help someone with less domain knowledge to still find the most relevant documents, whilst discovering more information than with a regular search.

The information discovery can be especially useful for legal authorities because a more targeted search can lead to faster and deeper insights. Furthermore, lawsuits that previously would be forgotten about can appear in the recommendations based on searches that were conducted in other cases. This discovery can further help those authorities to escape a ”filter bubble”, where either the lack of domain knowledge or not-well specified queries could lead to insufficient or low-quality results.

3.1.2. How does a recommender system work

A recommender system generally works by trying to find similarities between users to recommend users items that they usually would not have found. An excellent example of this would be YouTube. When a user searches for cat videos on YouTube, the users might be recommended more cat videos or, even, dog videos as the user might like animals in general. Recommender systems can be used a lot broader than just for videos. For examples, webshops use it to recommend items that the customer might be interested in buying, and dating apps use recommender systems to guess who would be a good match for the user.

3.1.3. Recommender performance metrics

A recommender system can be evaluated in various ways. A novel way to evaluate a recommender system is to split the available interaction profile into a training and test set. The training set contains

a percentage of the items a user interacted with, whilst the test set contains the other items. The recommender system is first trained on the training set and then asked to predict the rating a user would give for each item in the test set. Based on this rating and a scoring metric, a score is calculated. Possible scoring metrics are described in [11], the list below gives an overview of those described ratings, where P is the predicted rating and R the actual rating.

$$\text{Root Mean Squared Error} = \sqrt{\frac{\sum_{rating}(P-R)^2}{\#ratings}} \tag{3.1}$$

$$\text{Mean Absolute Error} = \frac{\sum_{rating}|P-R|}{\#ratings} \tag{3.2}$$

$$\text{Precision} = \frac{|\text{relevant item} \cap \text{retrieved item}|}{|\text{retrieved items}|} \tag{3.3}$$

$$\text{Recall} = \frac{|\text{relevant item} \cap \text{retrieved item}|}{|\text{relevant items}|} \tag{3.4}$$

The performance metrics can be further classified into three different measurement strategies, according to [10]:

- Offline: this is the easiest method of evaluation as it does not use user interaction. It is based on the ground truth from which the accuracy is measured.
- Online: a useful way to evaluate the recommender system. Using the Click-Through Rate (CTR) to evaluate the acceptance of recommendation in real-time.
- User study: a user study uses a questionnaire to evaluate the overall satisfaction of the system through ratings.

Picking the correct measurement strategy is essential, as the strategy greatly influences how precise the accuracy measurements reflect the system performance. While the offline measurement can be trivially implemented, it does not reflect system performance, as it does not take into account user behaviour like not buying lowly rated items, as is stated in [9]. The online system, on the other hand, takes this behaviour into account as it measures the recommender systems interaction in real-time in the production environment. This measurement system strikes the right balance between implementation cost and close system performance reflection; however, insights of the system-as-a-whole can not be derived from those results. The user study is the most thorough measurement technique, as the system will be evaluated by a questionnaire given to users, as is described in [10]. This measurement technique will give deep insights into how the recommender system works as a whole in an application from a user's perspective. This technique is, however, less frequently applied, as the success depends on various factors; such as questions asked, how many people fill in the survey or whether there is enough budget.

3.2. Different recommender systems

In this section, various types of recommender systems are discussed, giving insights into there strong and weak points, according to [10]. The techniques that will be discussed are *Collaborative Filtering methods*, *Content based methods* and *Hybrid methods* under these umbrella terms various possible implementations are discussed, allowing for a brief general overview of what is currently available.

3.2.1. Collaborative filtering techniques

Collaborative Filtering (CF) is an approach to recommend concealed items using information from other users. It computes a similarity matrix between all the users, usually using the K-nearest-neighbour (KNN) algorithm to cluster similar users. CF can be further categorised into three different approached: "Model based", "Memory based" and a "Hybrid approach". Those three techniques all have their advantages and disadvantages, and picking the right approach can significantly influence the performance of the CF recommender system.

3.2.2. Model based collaborative filtering

The model-based CF algorithm leverages a machine learning model to predict the ratings of items that are not rated before. Using a machine learning model boosts the performance of the recommendation over the memory-based approach, while also addressing the scalability and sparsity problem. However, creating such a model is not a trivial task. Furthermore, explaining recommendations generated by a model-based CF recommender system can be incredibly tricky. It's difficult to explain because it is hard to trace the recommendations through a non-deterministic model, as this model can be made from random processes.

3.2.3. Memory based collaborative filtering

The memory-based CF algorithm uses, user interaction profiles in order to recommended items based on some similarity function such as the Pearson correlation 3.5, cosine distance 3.6, Jaccard similarity 3.7, etc., where A and B are feature vectors. The creation of a memory-based collaborative system is, therefore, a trivial exertion, where information that is normally logged by a system can directly be used to create a recommender system. However, this type of recommender system faces a variety of issues, such as the cold start problem and data sparsity. The cold start problem arises when a new user starts to use the system, and no information is known about that user; therefore, no accurate recommendations can be made for this user. The sparsity problem often appears where the system contains few users-item interactions; this can lead to skewed recommendations or items not being recommended since some items never have any interaction.

$$r(A,B) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 (y_i - \bar{y})^2}} \tag{3.5}$$

$$cos(\theta) = \frac{\sum_{i=1}^{n} A_i \cdot B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{3.6}$$

$$J(A,B) = \frac{|A \cap B|}{|A \cup B} \tag{3.7}$$

3.2.4. Hybrid collaborative filtering

The hybrid based CF algorithm combines the memory-based CF with the model-based CF. This approach tries to overcome the constraints of the other methods, but this approach can drain serious programming and computing resources, as it is the most complicated CF approach.

3.2.5. Content based methods

This form of recommender system does not take other users into account but tailors its recommendations to a user's interactions. Once a user likes or interacts with an item, similar items to the interacted item will be recommended. These recommendations can lead to a lack of diversity in item recommendations as only those items that are similar to the interacted item can be recommended. Similarly to the memory-based CF, this algorithm faces the cold start problem, as users that did only interact with a few items will get poor recommendations. Furthermore, this method of recommendation requires a good understanding of users' preferences. On the other hand, the content-based method does not have to deal with the sparsity problem, where items lack reviews from users as those reviews are not taken into account for the content-based recommender system. Next to not having to deal with the sparsity problem, the content-based recommender system allows for easily explainable recommendations, which can lead to increased user satisfaction.

3.2.6. Hybrid methods

The hybrid recommender system tries to circumvent the limitations of both the content-based recommender system and the CF-based recommender system while improving the overall performance of the system as a whole. Multiple recommender systems are combined by applying a hybridisation technique. Standardised hybridisation techniques are well described in [2], in the list below a summary is given of those techniques.

- *Weighted* A weighing schema is determined between the recommender systems based on statistical information
- *Switching* A recommender system is used for particular recommendations while other recommender systems are used for different parts of the application.
- *Mixing* All recommendation from all systems are combined to one list of recommendations
- *Feature combination* Features from different data sources are combined in one algorithm.
- *Cascade* The recommendations from the first algorithm are refined by the second.
- *Feature augmentation* The output from the first recommender system is used as a feature for the second recommender system.
- *Meta-level* The model learned by the first recommender is used as input by the second.

Although the hybrid methods try to tackle issues like the cold start problem, they still arise as both techniques require user-interaction data. The cold start problem should be taken into consideration when applying a hybridisation approach for recommendations. A suitable combination of algorithms needs to be found to find the most optimal recommendations.

3.2.7. Issues with recommender systems

This section will re-iterate three well-known problems that will likely occur when creating a recommender system. The *cold start problem* occurs once a new user starts to interact with a recommender system or when a new item is added to the list of recommendable items. The problem occurs because recommender systems use information from the empty users'- items'- interaction matrix to recommend new items. The sparsity in the user-item-interaction matrix often causes the *sparsity problem*, where few items have been interacted with by the user. The *Flexibility problem*, which often occurs in CF based recommender systems, can lead to some items not being recommended. These items might not have been interacted with by any user or are recently added to the item list. When the recommender system is flexible, this could lead to non-specific item recommendations, while a rigid system could lead to new items never being recommended.

3.3. Improving recommender systems with implicit feedback

This chapter will explain Implicit Feedback (IF) and elaborate on how this technique improves a recommender system. The explanation is done by categorizing different techniques after which the improvement is explained. This research helps to back up the choice of which techniques will be used in this system which is done in chapter 3.5.

3.3.1. What is implicit feedback

A user uses an Information Retrieval (IR) system, like Bluetick, to find information from an 'anomalous state of knowledge'. The IR system transforms this search into a query which will approximate the information needed [14]. However, this approximation may fall short when the information that is needed is vague, incomplete or if the user is unfamiliar with the retrieval environment. Therefore, Relevance Feedback is used to automatically improve the system, which can be Explicit Feedback (EF) and Implicit Feedback (IF) [14]. EF is the feedback that is marked on purpose by the user, like marking a document as relevant. Implicit Feedback, on the other hand, is taking advantage of user behaviour to understand their interests and preferences better [5]. For example, when a user selects a specific object from a list, this could be seen as an expression of his binary preference. Additionally, could this preference infer a relationship between the selected object and the other visible objects in the list that are ignored [8].

One major convenience of collecting IF is that it requires no additional effort from the user of the system [8]. In the current recommender system of Bluetick, a suggestion is made on the explicit use of the 'is-relevant' button. By adding IF, a case could also be recommended based on more sophisticated feedback, like scrolling, mouse events or click-through rates. Due to the effortlessness of the user, the data can be collected in large quantities. One disadvantage is that the data is inherently noisy, messy, and hard to interpret [8].

3.3.2. Classification of the techniques

In this section, the different techniques are classified. In the paper by Diane Kelly and Jaime Teevan, this is done by two axes, called the *Behavior Category* and the *Minimum Scope* [5]. The Behavior is then split up into Examine (e.g. scroll, reading time), Retain (e.g. print, mark relevant), Reference (e.g. Copy-and-Paste), Annotate (e.g. Mark up) and Create (e.g. type). Each behaviour category can be examined at three different minimum scopes; Segment, Object and Class; however this can be ambiguous as a *bookmark* can be categorized in multiple scopes [5].

Based on Núñez-Valdéz [7], a list of examples of implicit metrics can be depicted.

- Duration of the search session
- Number of clicks
- Viewing time of a document
- Number of visits to a document
- Number of interactions with the document
- Number of copy-and-paste actions
- Scroll behaviour
- Marked as relevant
- Downloaded a document

One of the most promising techniques is time viewed at the content, as a longer view time indicates a greater user interest. Additionally, the number of visits is an indication that the user is more interested, as an interested user rereads these documents [7]. To get an accurate estimation of the view time, it might be helpful also to monitor scrolling behaviour.

3.3.3. How does implicit feedback improve a recommender system?

Implicit feedback can add value in two ways. Firstly, it can be used to improve the recommender system by creating a better ranking of the recommendations. Having the best recommendations on top is important in this system, and is called Learning to Rank. Learning to Rank is a core technology, consisting of two categories: Personalized Ranking with Implicit Feedback (PRIF) and Personalized Ranking with Explicit Feedback (PREF). The most used technique to achieve PRIF is Matrix Factorization using the Bayesian Personalized Ranking [6], which is most widely used and improved and will therefore most likely be a good start.

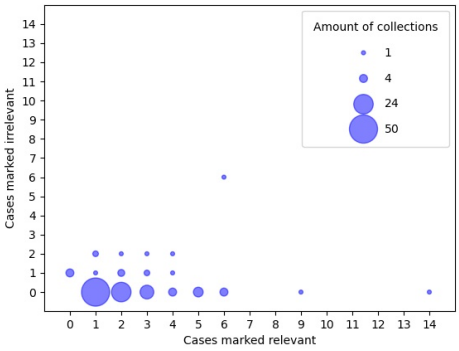
Secondly, the evaluation of the quality of the recommendations is always challenging, and the only metrics currently available are the relevant marker and the search history. While this gives a great starting point, it will not be enough feedback on what the user likes because sometimes the user is not reviewing their recommendation explicitly. IF will guess their preference and can give additional confidence on how relevant a recommendation was after it has been viewed/selected/marked [3]. Implicit feedback will provide more data to make better and extra recommendations, which will help the user towards their goal without them explicitly reviewing the recommendations

3.4. Data currently collected by Bluetick

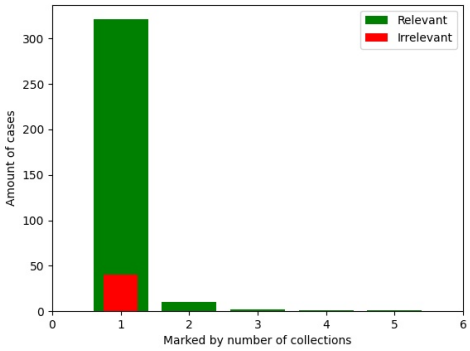
These sections contain an overview of all the user data being collected by Bluetick from July 2020 until the time of writing (November 2020). Accounts of Bluetick employees that are used for testing or development have been removed from the results. Please note that changes to the product have been made over the last couple of months that could have had an impact on the data.

3.4.1. User Profile

Bluetick provides new customers with several registration tokens. These can be used to create personal accounts for those that are going to use the search engine. A recent addition is that during the registration procedure, the user is asked about his profession and years of experience. Also, users can select several options to indicate their use for the search engine and how much time they usually spend on searching and looking at cases.



(a) Amount of cases marked per collection

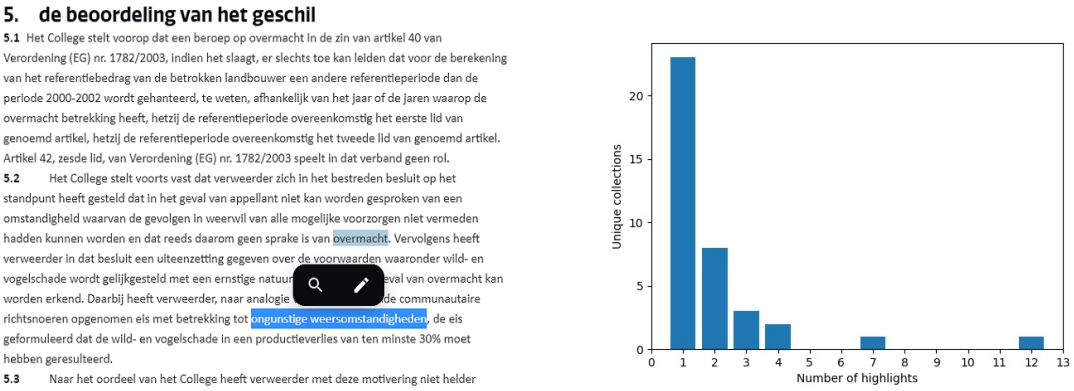


(b) Number of times cases are marked by different collections

Figure 3.1: Usage of marking feature in Bluetick

3.4.2. Marked cases

For each collection, the user can mark cases either relevant or irrelevant, or leave it unmarked. The user has access to an overview of all marked cases such that he can quickly look at them again. Figure 4.1 shows the usage statistics of the marking feature within collections. It can be observed that there are barely any cases that occur in more than a single collection. This sparsity raises the question of whether a user-based recommender system, where collections are considered users, would be advantageous.



(a) Highlighting feature in Bluetick

(b) Amount of highlights per collection

Figure 3.2: Usage of highlighting feature in Bluetick

3.4.3. Highlights

The user has the option to highlights pieces of text in a case document, as visible in Figure 3.2a. The user can retrieve a list of his highlighted pieces of text. From Figure 3.2b it can be observed that highlighting does not occur as frequently as marking cases in collections. This difference is because if a piece of text is highlighted, the case is automatically marked as 'relevant'. The lack of marking could also be partially explained by the fact that highlighting is a relatively new feature compared to being able to mark cases.

3.4.4. Linked from

There are different ways for a user to find a case. For example, users can click on cases that appear on the top of their keyword search, but they can also open a case document when it gets suggested to them from the recommendation page. Figure 4.3 contains an overview of how often a case gets opened from different locations in the application.

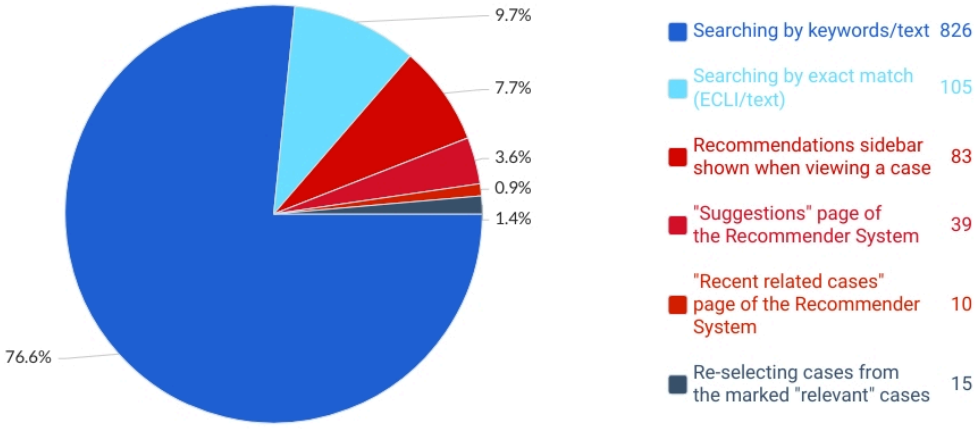


Figure 3.3: Pie chart showing how users found a case

3.5. Optimal setup for Bluetick

In this chapter, the recommendations are stated for Bluetick. Firstly, the recommendations are given on how to collect more data by exploring possibilities in implicit data collection. Secondly, the type of recommender system that Bluetick could use is given.

3.5.1. Using implicit feedback

To better analyse the quality of the recommendations, additional data in the form of user interaction should be collected. As discussed in chapter 3.3 the most promising techniques is time viewed, the number of visits and the scrolling behaviour. After this logging feature has been implemented, the correlation between these metrics and the relevant-marked documents can be analysed. This information can be used to find relationships. These relationships can then help define the weight of each metric and how it can be used in the quality of the recommendation.

3.5.2. What recommender system

Bluetick has a small user base and is expected to slowly grow based on customer research done by Bluetick. With over 600.000 cases and only around 30 users, the item space remains enormous when compared to the number of users. Currently, there is no see sufficient overlap in relevant cases between collections to make a user-based recommender system such that it benefits Bluetick in the near future. For this reason, the CF-based recommender system is not a viable approach, but a content-based recommender system could be viable. The content-based recommender system uses item's features in order to recommend similar cases and is therefore independent of user-item interactions. Furthermore, the explain-ability that comes with this approach can be valuable for Bluetick because it gives their customer greater insights into how recommendations are created for them.

3.6. Research Conclusion

Bluetick wants to offer its customers tools that are not available elsewhere. By adding a recommender system to their search engine, they provide value to their customers by decreasing the times it takes to search and by giving them more accurate results, even if the customers are looking for something that they cannot explicitly name.

To conclude this research report, the answers to the four research questions are given below:

RQ 1: Which types of recommender systems exist?

As mentioned in chapter 3.1, there are multiple types of recommender systems that all have their pros and cons. There is a collaborative filtering technique, which uses a similarity matrix to compute the similarity between users and between items to be able to recommend new content. Another method is content-based, where the recommendations are mainly based on the interaction of the user. Where the collaborative filtering is suited when a large amount of data is available, this is not needed for content-based recommender systems. Among this and other reasons, the team expects that a content-based recommender system is best suited for the project in the near future.

RQ 2: How can implicit feedback be used to improve a recommender system?

Recommender systems can be evaluated by implicit feedback, as is discussed in chapter 3.3. Using implicit feedback creates a more substantial data collection which consequently will result in more accurate recommendations. Implicit feedback is gathered by looking at the behaviour of the user instead of explicitly asking for feedback. As is explained in 3.5, time viewed, the number of visits and the scrolling behaviour are the most potential metrics that will be used.

RQ 3: Which data does Bluetick currently collect?

Bluetick does not have a lot of available data. The data that is currently being collected is described in chapter 3.4. This data comes down to the user profile data, marked cases, highlights and the linked-from data.

RQ 4: What kind of recommender system is best suited for this project?

The team provided a solution to Blueticks problem by choosing suited techniques, as described in chapter 3.5. The solution will be a content-based recommender system that uses implicit feedback and other evaluation metrics to improve the recommendations.

4

Problem Definition

This chapter contains a reformed problem definition and an explanation of why this was necessary. First, the team will delve into the situation of Bluetick after the research phase, then an oversight will be given of the new goals, and finally, the strategy to solve them is discussed.

4.1. The foundation for an improved Recommender System

After the research, the team delved into the code base of Bluetick. The team mapped the structure of their code in a class diagram. Analysing the code gave insights into the current state of the application. From analysing the currently available data and the state of the application, the team concluded that the application was not yet ready for the large-scale improvements discussed in the research section. The problems found during this exploration of the code are summarised in the list below.

- A small user base. In the research report, the team discussed the implementation of implicit feedback, which in turn would lead to an improvement in the recommendations. A small user base could lead to unstable recommendations since there can be a large variety in searches users do.
- Unstructured code. While the team wrote the research report, they thought that the recommender system was written with standardized computer science design principles in mind. The team discussed explainability as they wanted to explain to the user how a recommendation was created. The code structure was not suitable to create such explanations.
- No testing. From a software engineering perspective, testing is an important concept, that can help get confidence in the application. However, when the team started programming, it was obvious that no tests where created.

The before mentioned reasons lead all to believe that a redefinition of the BEP goals was necessary, as otherwise a high-quality result could not be ensured.

4.2. Updated problem definition

To guarantee a final product of high quality, the team made a new problem definition. After the research, this project's focus shifted towards better code quality instead of improving the recommender system. During the development phase, the team mostly wanted to focus on following the standardized design goals. The problem definition is divided into the following points:

- Scalability: This is important as start-ups tend to grow quite explosively. The team should make sure that the code can handle a large number of parallel users.
- Maintainability: This will lead to easily understandable and maintainable code. A maintainable codebase is one that could easily be tested. Within well-maintained code, it should be easy to solve bugs and other issues.

- Transparency: The system should increase the transparency towards Bluetick and the users of the application.
- Extensibility: This is important, as this will not be the last improvement done to the application. Therefore the team should create a stable foundation that can be extended easily.

The design goals are further distilled into a three-pillar plan of P1: Transparency, P2: Analysis and P3: Improve Recommender system. P1 mostly focuses on improving the traceability of the recommendations towards Bluetick and the user. Features that belong to this pillar are discussed in the refactor (5.1), the tests written after the refactor (5.2) and the recommendation explanations (5.3). P2 centres around the following features: more data collection (6.1), including weights in the recommender system (6.2) and the dashboard in section 6.3. Finally, P3 centers around improving the recommender system directly by implementing the warm start in section 7.1 and reordering in section 7.2. This pillared structure allowed for easy communication within the team and it gave insights into what a feature should tackle.

5

P1 - Transparency

The first pillar focuses on improving the transparency towards Bluetick and the user, as described in chapter 4. This chapter will describe the features that contributed the most to this goal. Each feature is split into three categories, design, implementation and evaluation. The design focuses on what decisions on an abstract level had to be made for that feature. The implementation section focuses on specific implementation details, and evaluation describes the review process of the specific contribution. The features discussed in this chapter are the refactoring, the testing and the transparency for the users.

5.1. Refactor

After analyzing the first version of the recommender system, it became quite clear for the team that the current structure would not be appropriate for expansion. An image of this system is shown in 5.1. The system already existed of different parts, but they were all used in the same functions, without clear distinction. Not splitting the code made it hard to find bugs, test, and maintain the old code. After discussions with the TU Delft coach, Bluetick, the team decided to refactor the old recommender system. The functionality would remain mostly the same, but the approach would be different. The team made a plan to split up all the parts with an eye on the product's future. Code maintainability, stability and future expansions were the main goals of this refactor.

5.1.1. Design

The design decisions that had to be made for the refactor only applied to the back-end code. For clarity, the decisions have been split into Data transfer object (DTO) and recommender system interfaces. The classes and interfaces were designed in such a manner that they would adhere to the Single Responsibility Principle as much as possible.

Data transfer objects

The team decided to split the recommendation data logic into classes. Two classes form the backbone of data handling: The ScoreStore and SplitScore. The system uses these classes to handle all data that is in memory. Functions were written for each of these classes, to do computation or actions on the recommendations: this improved code maintainability and decreased code duplication. An image of the class diagram can be seen in 5.2.

Recommender system interfaces

For the recommender system itself, the team chose an interface design pattern [12]. The team split up the recommender system's partial functions into separate partial systems, which each have their own database table and logic. However, they all have the same base functionality. Splitting up the systems made them consistent and made sure they worked roughly in the same way, but it still allows for different approaches where needed. By using an interface, it also makes it very easy to add new partial recommender systems. The scaleability was a larger reason why the team chose to use an interface design pattern. An essential part of the interface is that it allows for straightforward addition

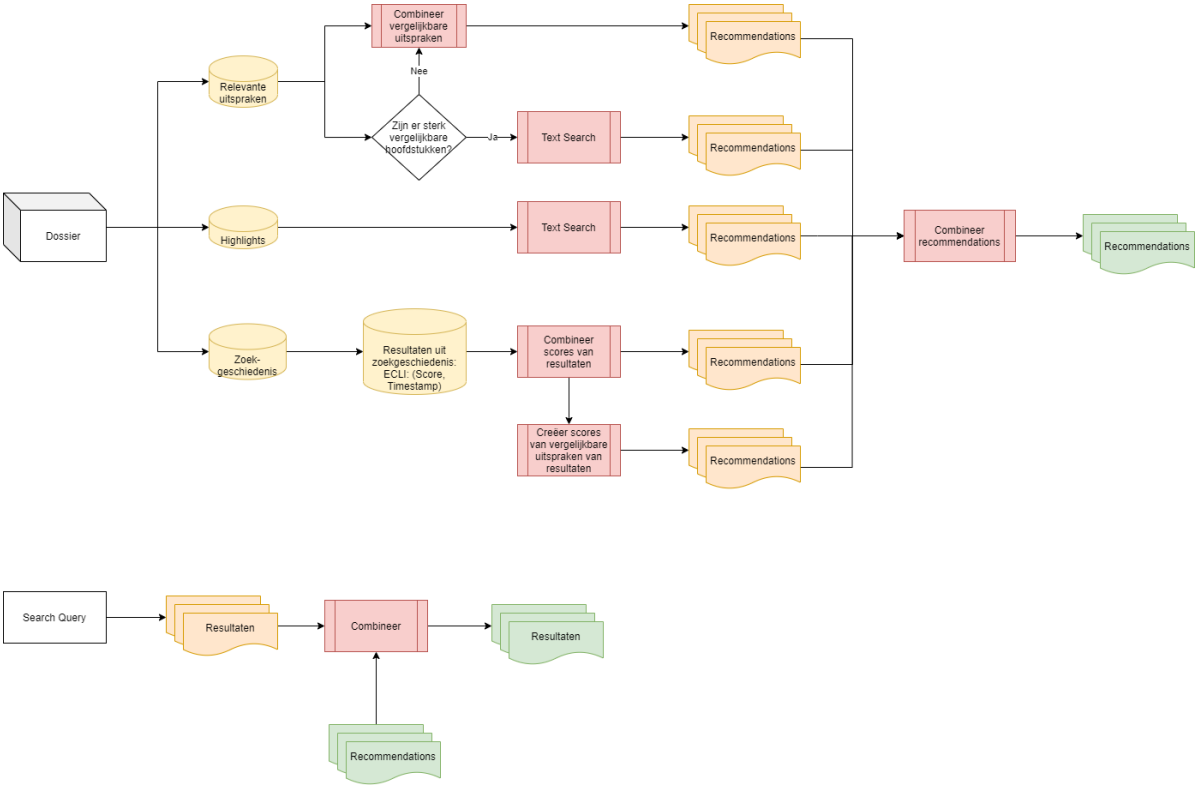


Figure 5.1: Workflow of the old recommender system

of new systems. In case Bluetick wants to expand the system, they can add a new part in an orderly and maintainable way.

5.1.2. Implementation

In this section, the implementation is described. Because this consists of the Data transfer object and the interface, this section will also elaborate on these parts.

Data transfer objects

A ScoreStore acts like a dictionary, where the key is an ECLI. The data is information for that ECLI: Specifically a SplitScore (more information in the next paragraph) and information about the ECLI's features. Therefore, a ScoreStore instance holds a list of ECLI's and a score, used as recommendations which are sent to the front-end and displayed as a list of recommendations. The ScoreStore object contains functions for computation and actions on the recommender system, like combining ScoreStores from different partial systems, multiplying all scores by a factor, adding new entries, and more.

A SplitScore is a more simple object, containing two parts: a list of scores together with their reason, and a total score, which is the sum of all separate scores. This way, the total score can easily be retrieved, but the partial reasons and their importance is also easy to get. SplitScore contains some functions as well, mostly for adding new partial scores and multiplication. Both classes can be seen in 5.2, together with their most-used functions.

Recommender system interfaces

For the interface, the team set up a few essential functions. At the very least, a partial recommender system must retrieve its recommendations, update the recommendations with a new entry, and in some partial systems have a remove function, for when, for example, a case marked as relevant is un-marked. These are the interface's functions; however, the partial systems are free to implement helper functions. The partial recommender systems currently implemented all work in the same way: Part one contains the raw entries. For example, in the partial system that handles cases marked a relevant, there is a list of similar cases compared to the marked relevant cases. This list can

contain duplicates, even cases already marked as relevant. Then there is the second part, with a precomputed list of recommendations. Here, a function adds the scores together, so there are no duplicates. Precomputing the recommendations increases the speed of the system drastically.

The old code has been split into six partial systems based on: relevant cases, highlights, keyword searches, ECLI searches, text searches and law articles given in the description of a collection. These systems operate apart from each other and are entirely separate. They all have a list of raw entries and a list of precomputed recommendations. Relevant cases and highlights are very similar, so are all the searches. Only the law article system has nothing that looks like it. How the systems interact with each other can be seen in Figure 5.3.

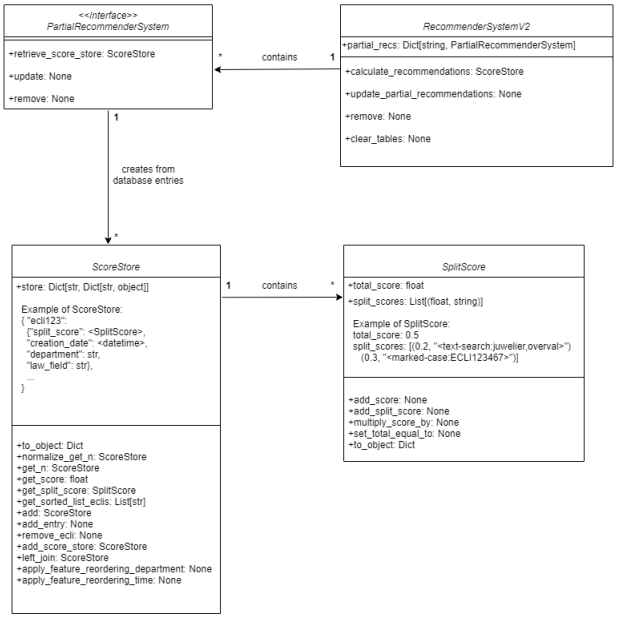


Figure 5.2: Class diagram of the new version of the Recommender System

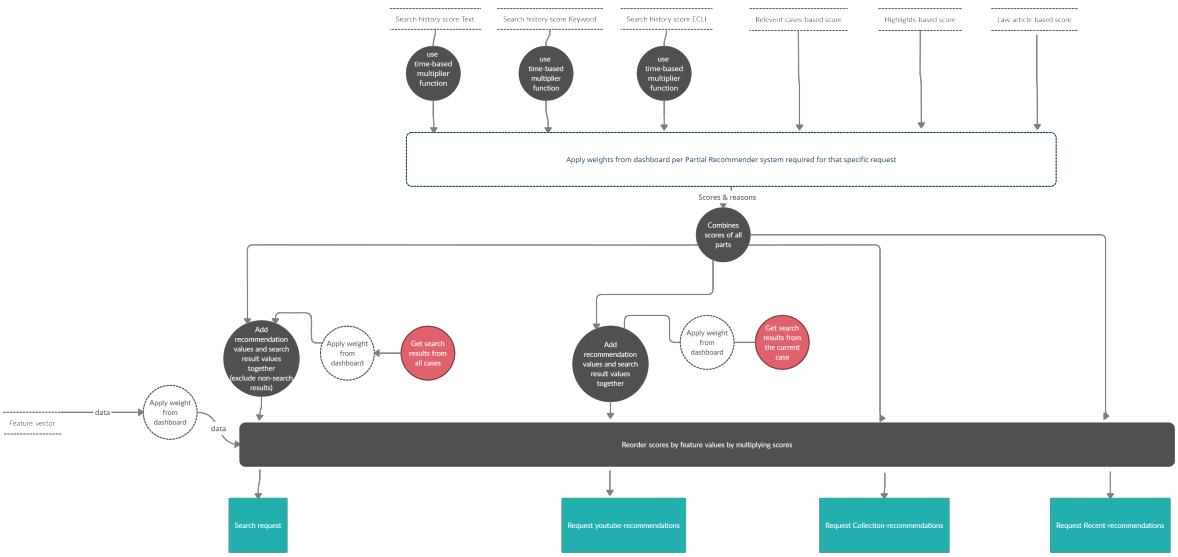


Figure 5.3: Workflow of the new recommender system

5.1.3. Evaluation

Different aspects can be taken into account when evaluating the refactor performed in this project. First of all, and as mentioned in the design part, the intention was to make it easier for Bluetick to maintain the system and make changes on the go. When wanting to introduce a new sub-recommender system, Bluetick can now create a class for it and inherit the functions from the interface.

Because the new design, the code has become less complicated. Functions are better split up, there is less code duplication, and by giving classes a single responsibility, the code is easier to understand. The team also added extensive documentation and type-hinting. The refactor also increases testability drastically. While refactoring, the team created many new tests for separate functions. Creating tests was almost impossible because most of the functions interacted with the database and were very large. Now that the team has split everything up, it is more simplistic to test separate functions instead of single functions with huge functionalities.

5.2. Testing

Having a well-defined testing strategy is vital to any company. However, automation in the testing strategy varies significantly. In the beginning, there were little to no automated tests. Furthermore, there was no clear testing strategy. This led the team to believe that quality improvement could be achieved by setting up various test runners for Bluetick.

5.2.1. Design

Bluetick lives in a so-called mono-repository where the entire application, as the name suggests, is contained within a single repository. The Bluetick application exists of a react front-end and two API services. This has various benefits of which one is the ease of testing. Software testing generally considers three levels of testing, namely unit, integration and end to end tests. To help Bluetick get more insights into why and mostly how software needs to be tested, tests were created on all three levels. For both API's test packages were created that mirrored their respective application structure, thereby ensuring that the test's goal could be observed with one glance. Additionally, two coverage tools were installed that can measure code coverage on various metrics. The coverage tools allow Bluetick to get a transparent overview of what still needs to be tested and where further test improvements can be made.

5.2.2. Implementation

This section will discuss how the team implemented the testing strategy for Bluetick. This will be done by going up the testing pyramid for all services within Bluetick.

The unit test followed mostly a strategy of using test doubles, where in-memory databases and mocks were used to isolate a component. This isolation strategy is necessary as unit-tests should be fast, reliable and easy-to-understand, however too much isolation leads to meaningless test as a component never exists alone in an application. Therefore it is crucial to focus on getting meaningful levels of isolation. This is achieved by mocking the responses from other services, or non-deterministic elements of the application, such as machine learning algorithms. Those mocks give the team fine-grained control over the components' behaviour, which in turn leads to easy to write and understand test code. An additional benefit of writing tests is that other developers can often look at the tests to understand what the code is supposed to achieve, thereby reducing the time necessary to understand each others code.

The integration test lives mostly at the boundary between different services, such as the Postgres database, MongoDB and the two API services. As the name suggests, the integration tests focus on testing the interaction between different services. These tests are harder to set up but give a higher confidence level that the application is working according to specifications.

The end to end tests test the most significant paths through the application, ensuring that most of the paths are working correctly. These tests are often flaky, meaning that the test can fail without an apparent reason, but give the highest confidence that the application works properly. Since the Cypress test allows each to write and maintain end to end tests, the test could be considered more like a hybrid between integration and end to end tests.

5.2.3. Evaluation

The tests were successfully added to the application and had a significant impact on the code quality, furthermore, this lead to Bluetick being more interested in writing high-quality code. Finally, the list of steps that Bluetick employees would generally run through when releasing a new version of the application can now be fully automated. This can save time and improve consistency between releases.

5.3. Transparency for users

At the start of the project, there was a lack of transparency towards the user, which is defined as the amount of information given to them about the recommendation source. The source means which part of data was the most responsible for giving that particular recommendation. The goal of giving more transparency to users was to improve the understanding of the search engine. The hypothesis was that the more the user knows about the source, the more the user will feed the application with useful information beneficial for the recommender system. Additionally, as described by Julie Daher [4], explaining recommendations makes it easier for the user to make decisions and leads to a higher trust in the system. More concrete, the hypothesis was that the user would interact more with the system and give better search keywords. This was also the guess of the User Experience (UX) designer and by the client. Therefore the team decided to design, implement and evaluate this feature.

5.3.1. Design

There are a few different places where recommendations were given, and therefore a design must fit all these different locations for it to be consistent. Firstly, a small design was created by the team to give an impression of the final result. This first version included an additional tag at the top, where the information was given as displayed in figure 7.1. This version was sent to the UX designer, and a meeting was scheduled to further brainstorm on how this should be implemented. When the design was finished, it was time to implement this feature.

Figure 5.4: First design of the user transparency

5.3.2. Implementation

The refactor made it possible to have more information about the reason for the recommendation. This enabled the frontend to display the source of the recommendation. For each type of recommendation, a different number of reasons were needed to be shown; therefore, this was one of the variables. Together with the reasons, which were structured in a way `<type-information>` e.g. `<keywords-keyword1, keyword2>`, which were parsed at the frontend into a user-friendly sentence like *You got this recommendation because you searched on keyword1 and keyword2.*

The final design was different from the first version. As shown in Figure 5.5, the reason is shown beneath the dossier instead of the tag on top. This was done to prevent the design from being very cluttered and because unimportant information should not draw too much attention. One of the features

Figure 5.5: Final design of the user transparency

of the reason display is that the ECLI is clickable. This makes it possible for the user to navigate to the source of the recommendation. The client liked the display of the recommendation source so much that it was also implemented into the search results, which was a big compliment for the team.

5.3.3. Evaluation

For this evaluation, the team contacted two lawyers and asked them about this feature. The main goal was to evaluate whether this feature was helpful for the user and to what extent it could positively change the user's behaviour.

Firstly, the feature was evaluated very positively by the two subjects. Both subjects claimed that it was a handy feature to see where the recommendations are coming from. Mostly because they like to know how certain things work instead of them being a black box. Secondly, they liked that it indicates why certain documents might be similar and therefore, relevant. If the reason already stated that it had been recommended because of a specific law article, it is a great help for the lawyer because they now know where to look for inside the recommended case. Finally, the place and design were reviewed as positive as is does not draw too much attention, but it is there when the user wants to read it.

5.4. Conclusion

By introducing the features described in this chapter, the team believes that the transparency for Bluetick and the user has improved. The main contributions were, the refactor, thorough testing and the recommendation explanation. All those features contribute to the entire transparency model that was envisioned at the beginning of the project and was described in Chapter 4.

6

P2 - Analysis

The second pillar focuses on improving the analysis possibility for Bluetick, as described in Chapter 4. The analysis will focus on gathering more data, having the means to visualize the data, and adjusting the system based on the conclusion drawn from the visualization. In this chapter, the data collection, combining strategy and the dashboard will be discussed, each similarly divided as done in Chapter 5.

6.1. Data Collection

In the old system, there was hardly any data collection. The only data collected was which user viewed which case and the necessary things such as saving cases that a user marked as relevant. Additionally, the path someone took to open a case (e.g., from the suggestion page or a search) was also already being logged. To gain more insight into the product's usage, the team believed it to be helpful to log more and different data. This data could then also be used to measure the effectiveness of the recommender system. Therefore, the idea arose to log the user activity when reading a case (e.g., the reading time) and the opened cases' law area. With this extra data, the team hoped to give Bluetick more insights in the recommendations and user behaviour.

6.1.1. Design

For user activity tracking, it was necessary to ask the user for permission. Therefore, a new checkbox was needed in the front-end, where the user fills in if he permits cookies and tracking or not. The checkbox was added in coordination with Bluetick so that it would fit the rest of the design. The team decided that the activity logger belongs in the front-end, in a separate file.

6.1.2. Implementation

The implementation was done in two different parts. First, the team wanted to collect data for the recommender system, even if the data was not yet used. Therefore, the team made a JavaScript file that tracked the time a user would actively spend on a page, how far the user scrolled/read the page, and then sent it to the server's back-end. The back-end would start saving the collection number together with the gathered data.

The team also created a logger to see how often the recommendations were retrieved. This logger was essential to see how often users would use the recommendations instead of using the search function. This logger was simpler to build, as it only required code in the back-end. The data was then also stored in the back-end, in the database.

The law area data collection focuses on tracking user interaction with regards to the documents they open. The law field of the documents are tracked when the user clicks on them and after a certain threshold is reached with a collection, a soft filter would be done on this law field. The soft filter would help the recommender system to find better recommendations for the user.

6.1.3. Evaluation

When working on using the user's activity to improve the recommender system, the coach pointed out that this path did not have much potential. The team took this advice and ceased working on those

plans. The activity logger and data saving have been finished. With the activity logger and data saving finished, the data gathering began. However, the team never came to processing the data. The idea was, that when the recommender system was ready, the team could see the differences in how users made use of the recommender system, and how active they were in the recommended pages. Because the team never proceeded, the data is currently only being logged, and not being processed. Valuable time was spent on other parts of the system instead. Thanks to the coach, the team worked on parts higher important parts, resulting in completing the second version of the recommender system. For the future, the logger and activity tracker are still in the code, still gathering data, which are available to compare different versions of the recommender system. Another possibility for the activity tracker is to be used in the recommender system, by calculating a multiplier based on how much interest a user had in a case and then applying it to the recommendations based on the case the user saw.

The law area data collection needs further work. However, a first version, where data is collected, has been pushed to production. This collection will allow Bluetick to create a model that helps the user to get more personalized recommendations.

6.2. Including weights and combining partial systems

As stated in section 5.1, the new recommender system is separated into multiple *partial* recommender systems. Each system has its own set of recommendations and therefore, needed to be merged or combined in some way to give a final list of recommendations. Conclusively, a system where each partial system has its own weight solves this problem of merging. The system uses a normalization technique to bring all separate values into the same range.

6.2.1. Design

To make a difference in the importance of the different systems, weights needed to be used. These weights could be different for each recommendation type. In the application, there are three different types; a collection recommendation, a document recommendation and recent recommendations. Every type has a different set of weights to give more priority to one *partial* system. The recommendation types are described in more detail in appendix E and screenshots of the front-end are displayed. The importance of each subsystem is enforced by multiplying the weight with scores.

Normalization is a technique often used to normalize numerical values into standard range. Normalization is needed because it ensures that any two *partial* recommender systems with equal weights have the same influence over the final recommendations. The reason for this is that the current implementations of *partial* recommender systems differ in the way they scale scores; in some the score values returned by the partial recommender system are relatively low compared to others, even though they received similar amount of updates.

6.2.2. Implementation

The team decided to save the weights into the Postgres database of the API. This made it easy to review, add and adjust the weights according to the needs. However, the weights are also used in the search engine, and therefore these values are added to the request to the search and used there. If no weights are found, default values at the search engine are used, and a warning is raised for the developers. At each *partial* system, the weights are then used to alter the individual scores.

For the normalisation, different algorithms could have been used by the team to normalise the scores. The most notable difference between them is the extent to which they maintain the distance between data points. However, due to this project's scope, the team could not perform very detailed research into the best algorithm for this particular use case. Therefore, the team decided to look into a few papers that analysed different normalisation algorithms and implement one that looked most promising. In [13] it was observed that the Ranksum algorithm, explicitly defined in equation 6.1, resulted in the most accurate score predictions for their feature set. Therefore, it was decided to implement the same algorithm and allow for Bluetick to make changes to this in the future easily.

$R_i : rank$ by feature i in which the item with the highest feature value f_i gets a lowest R_i

$$normRank(i) = \frac{MAXRANK - R_i}{MAXRANk} \tag{6.1}$$

6.2.3. Evaluation

Currently, the weights are stored at a different place than where they are used. This implementation is somewhat counter-intuitive. So while this was the best option for the current situation at Bluetick, it is desirable to make it more intuitive by changing this in the future. Apart from this, the code is relatively easy to understand and can be easily extended by more weights. Additionally, it can be concluded that it is hard to see the actual effect the weights have on the recommendations. More information should be gained in the future to get a better understanding of the effect. Also, there is still some room for improvement at the loading of the weights into the database. No default CSV can be loaded, which results in the manual creation of the weights. This could have been done better.

Due to limitations described in Section 6.2.2, it was not possible to verify the Ranksum normalization algorithm's performance. However, choosing this algorithm was not done on a completely arbitrary basis; the team considered it as the most promising candidate.

6.3. Dashboard

At the start of the project, little to no quantitative information was available to Bluetick regarding the recommender system's performance. With that in mind, the team decided to implement a dashboard. The dashboard's main goal was to increase the data analysis possibility for Bluetick, as described in Chapter 4.

6.3.1. Design

Before working on the dashboard, the team created some mock-ups of visualizations (figure 6.1) that ideally should be included. Since the team briefly went over these mock-ups with the Client and adjusted them after receiving feedback, they already had an idea of what kind of visualizations would prove valuable and thus saved some time by limiting the time spent on redundant parts.

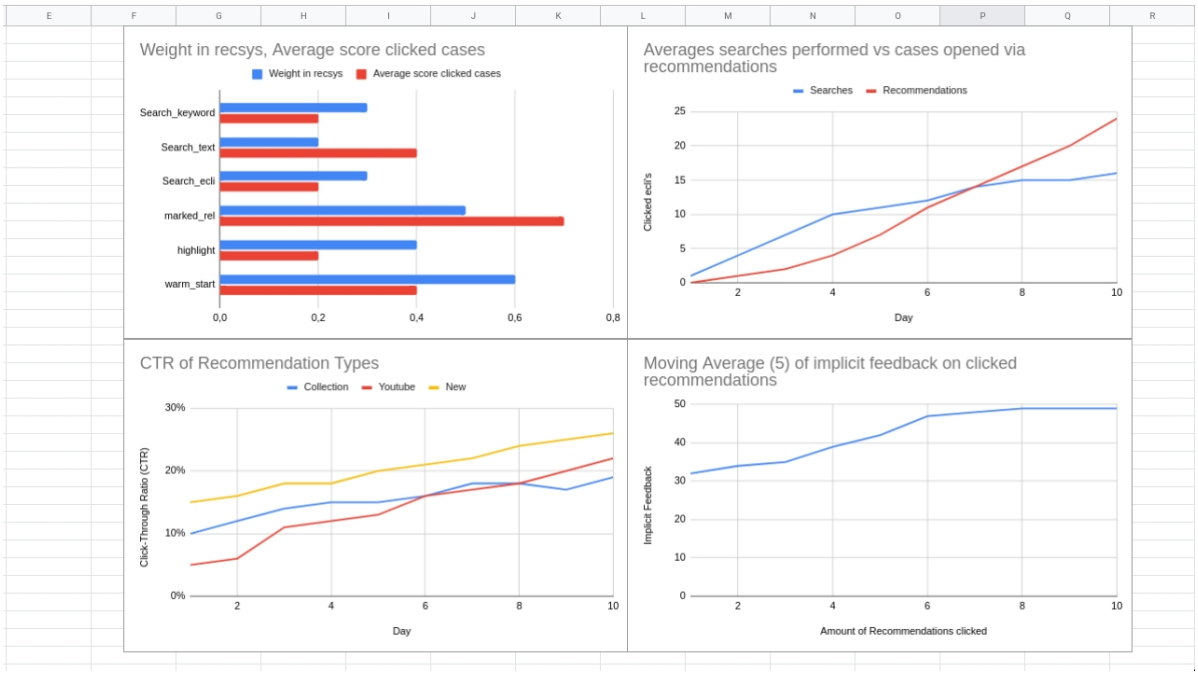


Figure 6.1: Mock-ups used for the recommender system dashboard

6.3.2. Implementation

As described in Section 3.4, much user data was already collected by Bluetick that could be used to provide useful insight into the usage of recommendations. However, the team and the client also wanted to display the Click-Through Rate (CTR), which was not yet possible due to the lack of data about the number of times recommendations were fetched from the system. Therefore, the team created a new table were the amount of calls to the recommender system is logged, as is described in section 6.1.

Regarding the platform used to host the dashboard, the team decided to use Metabase, a business intelligence tool already in use by Bluetick. The main issues with this tool for the team's purpose were its limitation to SQL queries and its limited visualization capacity, making it more challenging to create some of the visualizations. However, the team believed that its benefits outweighed its downsides. First of all, Metabase was easy to use, extend and maintain, and thus it already adhered to most of the design goal pillars as described in Chapter 4. Secondly, Metabase was already a familiar tool for most of the maintainers of Bluetick.

Not all of the visualizations, in the end, matched their mock-up. This could be mainly attributed to some of the limitations of Metabase and the fact that working with the data for a longer time resulted in some new insights in better ways to display the data. Due to a lack of time, the team could not create visualizations that used recommender system scores. As can be seen in Figure 6.2, the visualisations that were implemented are limited to the usage of recommendations and not the way they are composed.

Figure 6.2: Final implementation of the recommender system dashboard

6.3.3. Evaluation

As described in section 6.3.1, the team already worked with mock-ups for the visualizations and received feedback on them from the maintainers of Bluetick. However, during the implementation phase of the dashboard, some slight changes to these were made without intermediate review (Section 6.3.2). Therefore, the team decided to have a second round of feedback on the dashboard by the same client and made slight changes according to the feedback received.

6.4. Conclusion

This chapter discussed the improved data collection and the combining strategy that were implemented for P2: Analysis. The team believes that a great effort has been made for Bluetick to start analysing the recommender system properly. In the future, Bluetick can combine the implicit feedback collected in 6.1 with other data they have collected and improve the recommender system by tweaking the weights that were described in 6.2.

7

P3 - Recommender system improvements

In this chapter the improvement that the team made for the recommender system are discussed. First the warm start will be introduced, after which the reordering of results is discussed. This chapter will be similarly divided as Chapter 5.

7.1. Warm start

Recommender systems only work when data is available. Bluetick encourages its users to create a new collection for each new case, thereby organising case-relevant information in a structured manner. An added benefit of this is that the recommender system can personalise recommendations on a collection base. This does however lead to the problem that for each newly created collection the cold start problem arises, as described in section 3.2.7. To combat this issue, the team implemented a feature that allows users to initialise a collection with additional information.

7.1.1. Design

A lawyer often has a general idea of what they want to search due to experience, general knowledge, or searching outside of Bluetick's product. Therefore, the team designed a feature that would capture this information, as it can help to reduce the cold start problem. A small meeting was organised with the team and the User Experience (UX) designer to discuss how the UI should be changed to accommodate this feature. From this meeting, the team concluded that the information should be asked subtly.

This feature was designed to improve the quality of recommendations by reducing the severity of the cold start problem; however, this leads to added complexity to the recommender. Therefore the primary design goals should still be considered when the warm start is implemented. Firstly, the warm start should be written in a maintainable way, such that Bluetick can find bugs quickly, and code can be changed easily. Secondly, the warm start should be scalable, thereby not having a tremendous impact on the application's responsiveness. Finally, the warm start should be traceable through the application for Bluetick and the user.

This feature focuses on improving the recommender system by adding new input options for the user. This leads to an improved user experience and more relevant recommendations.

7.1.2. Implementation

From the design phase, the team decided that the warm start would focus on active data collection, which was successfully implemented. The active data collection focuses on getting a user to add law-articles to their collection, as shown in figure 7.1, which would initialize the recommender system with suggestions related to those law-articles. Next to having an idea of what the user is interested in, the system can also start giving recommendations when the collection is created. These suggestions actively reduce the cold start problem.

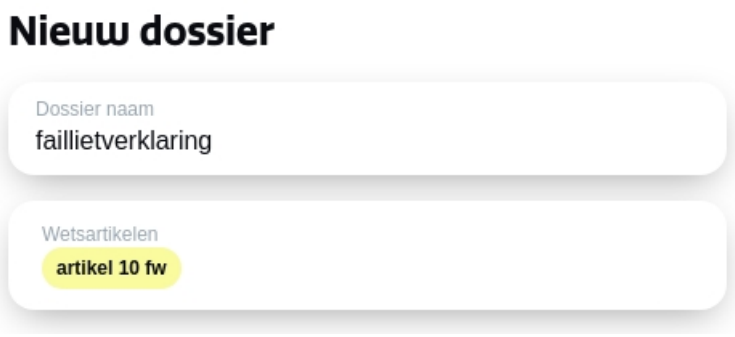


Figure 7.1: Final design of the active warm start

7.1.3. Evaluation

The active data collection has successfully made it to the production environment. It allows users to create collections with law articles, which give them a feeling of control over the recommender system. Furthermore, this system has been extensively tested through unit, integration and e2e tests. It successfully improves the recommender system's usability by giving the user more control over the recommender system. Bluetick also came to the same conclusion. Finally, the team evaluated this feature with two subjects that work directly in the law area. The feature was positively received by both of them, stating that it can be advantageous as a soft filter. However, the feature is not user-friendly enough as it is not giving feedback on when the system accepts a law-article. Furthermore, not all lawyers will know what law-articles they are interested in from the beginning of the case. Therefore it would be helpful if the user is reminded that law-articles can be explicitly added to the recommender system.

7.2. Reordering results

Another feature requested by Bluetick is the ability to reorder the results from the recommendation system. For example, a recent (less than three weeks old) case is more important to the user than a more than three-year-old case. However, unlike a regular search, where a user would be able to sort all the results on the date, this is not useful in the recommendations. Instead of sorting the recommendations on data, a reordering is applied. This reordering is done so that the underlying structure of the recommendations perseveres. This is necessary since a sort would rearrange recommendations based on dates, not considering the actual score a recommendation had. Therefore a soft-reordering was applied. This reordering would boost a recent case to a higher place while taking into account the original score that the recommender system calculated.

The criteria for the soft reordering were discussed in a meeting with Bluetick. It was concluded that the following criteria should be applied in the reordering: the date and the court belonging to the case. A more recent case is more important, as it can over-rule older cases. The same goes for a higher court: A higher court can over-rule a lower court.

7.2.1. Design

Due to the extensive refactor as was described in section 5.1, it was not necessary to change neither the interface nor the classes of the recommender system. The design of the recommender system was able to support the reordering natively. Thereby further reinforcing how important the refactor was, for the new functionality that was added.

7.2.2. Implementation

The reordering feature required two new implementations. First of all, the reordering features for each case need to be retrieved from the system. The team decided to refactor the ScoreStore class and decided to add extra fields for the reordering. Each feature could then be added together with the case to the database.

The second part of this feature was trivial to implement. After storing the features in the database,

the next stop was to multiply the score that the recommender system assigned to each case by a multiplier, influenced by the features and the weight assigned in the admin dashboard. The recommender system now multiplies the scores of each case by their features and weights. For the date, the team made a function that boosts cases that are less than three weeks old, with a value from a settings file used as a power over the resulting multiplier. There is a dictionary in the settings file for the court weights, where different multipliers can be supplied for each court.

By multiplying the scores of the cases, the order of the cases is also changed. The change of order is because of their score in the recommendations that orders all the cases.

7.2.3. Evaluation

The approach the team took required a refactor of the ScoreStore class, which was not optimal. It also made the system a lot slower, as now data had to be collected about the case's date and court. This information has to be retrieved from the external system, sometimes many times, causing the processing of new recommendations to be a lot slower. The slow speed is especially evident in the Youtube-style recommendations. In these recommendations, an ECLI search is done first on the opened case, and then the data is processed, then the recommendations are computed, and only then are they shown to the user. The Youtube-recommendation is the only time the user will see the recommendation system's long loading and processing time.

The approach of the team, therefore, took more time than necessary. The team should have planned for this earlier when they did not create the ScoreStore class yet. The structure also does not allow for missing information, because it causes the recommender system not to produce recommendations. Later on, this could be improved by Bluetick. A positive side of this refactor, is that it is very straightforward to add more fields. These new fields can then easily be re-ordered in the recommender system.

7.3. Conclusion

The team believes that the added functionality will help Bluetick in the future, as the new features are closely related to what was already on Bluetick's future development plans. Although both features are not fully implemented, the team thinks that a solid foundation has been made for each of them and hopes that Bluetick will further improve upon them.

8

Process

In this chapter, the development process will be described. To stay on schedule and up-to-date with each other, different tools have been used during the project. Additionally, different meetings structures have been used to create consistent communication between all parties involved. To further explain the process, it is divided into communication, meetings, the scrum methodology, pull-based development. After that, the SIG feedback is presented to give an overview of the software quality. Lastly, the team reflects on the team members and the process.

8.1. Communication

Communication is always a key part of the process. Between team members, it is essential to know which features are implemented by whom. Furthermore, the client plays an important role in the problem definition and product feedback and should often be talked to. Additionally, it is important to discuss problems that arise as quickly as possible, not only with the team but also with the coach to not further delay progress. Due to the external factors, a global pandemic, the team could not go to the office, so all meetings were done online.

8.1.1. Team

The team met every day, often twice a day. During these meetings, progress and problems were discussed. During the first few weeks, these meetings included multiple presentations to keep the rest of the team informed about new discoveries of the codebase. This helped the team to reduce the learning curve instead of reinventing the wheel individually. More towards the end, these meetings also included demos of new features to gain feedback from the team. These meetings were done via Discord, which also included different channels based on the different parts of the project, including large branches, logistics, research and off-topic channels.

8.1.2. Client

Every Tuesday, a meeting was scheduled with the client. During these meetings, the team tried to ensure that they understood the problem at hand correctly. When the research was done, the team reported back to the client what the possibilities were and presented a plan to tackle the problem. Lastly, when a feature was implemented, the team showed this to the client and asked for additional feedback to make sure the feature fit the client's original needs. Additionally, one of the team members attended the daily stand up with all employees of Bluetick. This made sure that the team was up to date with the company's most recent pursuits and made sure that other employees of Bluetick were up to date with the progress of the team.

8.1.3. Coach

Every Thursday morning, a meeting was planned with the coach. In this session, the progress was presented to the coach, and additional ideas and feedback were given by the coach. This weekly basis helped the team as the coach was well informed about the project's progress and problems.

Consequently was the feedback of high quality and all problems that arose was addressed quickly. This guidance was a great addition to the productivity and focus of the project.

8.2. Meetings

For each meeting, various preparations were made. This included stating an agenda, which was send a day in advance so everybody could prepare their part. This made sure that all participants were on the same page before the meeting started, which significantly improved the meeting's efficiency. Additionally, at the start of the meeting, the agenda was discussed to ensure that no crucial points were missed. This proactive attitude of the team was evaluated as very positive by others. All meetings were every week at the same time, which made the week very structured. All team members enjoyed this structure.

8.3. Scrum methodology

For this project, an agile methodology is used to plan, design, implement and evaluate product and process. Scrum allowed for a flexible workflow where new problems are easily planned into the next sprint. After the research was done, it was concluded that some of the earlier ideas were not feasible, and therefore, the team should shift their planning and design to make it more feasible.

8.3.1. Sprints

Because the project's focus shifted quite often in the start, the team decided to stick to weekly sprints. This enabled flexibility, which made sure their shift of focus could be done quickly. Each sprint consisted of a sprint planning on Monday morning, daily stand-ups, daily stand-downs, a sprint review and a sprint retrospective.

The daily stand-ups made sure the team was up to date with each other progress, and there was room for feedback if somebody needed that. These meetings could take quite long in the first few weeks, as a lot needed to be discussed and decided. However, after these weeks, the stand up took no more than 15 minutes before everybody got to work. The daily stand-down was mostly to prepare upcoming meetings and show the progress that was made during the day. This made sure all team members also saw what the others were doing.

The sprint retrospective evaluated the process, including the meetings, communication, issue definitions, merge-request and more. Every week, the team evaluated based on these categories what the team should 'start doing', 'keep doing' and 'stop doing'. Every review, the team looked back and the older reviews if the feedback was adopted correctly. Additionally, this review included a round of personal feedback. This feedback tries to keep the team together and resolve any conflicts that might come up as soon as possible. Giving constructive personal feedback and compliments improved the overall teamwork and positive vibe. It also created an open environment where all team members could say it if something was on their mind.

The sprint review was a structured meeting to look back at the product backlog and see which progress is made. This was the moment to reflect on the priority of the issues and whether the team is still on track. The progress was then shared with the client in the upcoming meeting to also communicate the product's current state.

8.3.2. Backlog

For the general product backlog, Trello was used. Trello allowed the team to describe each part of the problem into smaller parts. Each of these parts was then constructed into concrete issues which should be implemented in the sprint. This sprint backlog contained all issues, where each issue must contain the corresponding labels, priority, checklist, description and assignee. The checklist was vital because it forced the team to further breakup issues into concrete steps to complete the original issue. Additionally, others could see very detailed information about the current state was. Together with the description, this gave a very nice overview of which goal was met and the definition of done and the goal. Also, Trello made it easy to keep track of the reported bugs, problems that have come up and more.

8.4. Pull based development

For this project, pull-based development is used. Every feature or bug gets their own branch on which it is implemented. When the feature is done, it gets reviewed by at least two team members which means giving feedback on code quality and running the application locally. All will be done before it is merged into the master branch. In this specific case, the team had their own master branch called `bep-recsys`, which was then merged into the master of Bluetick. The team made sure that bep-recsys was up to date with the master of Bluetick. The Master branch is then deployed to the staging environment, which is tested by other Bluetick employees. This feedback will be reported back to the team and was fixed as soon as possible. When everything works as expected, the feature gets deployed to the production environment.

8.5. SIG feedback

At the end of week 6 and 8, the code must be submitted to the Software Improvement Group (SIG). This is an assessment of the code quality. Metrics such as code duplication, unit complexity, module coupling and more are used to qualify how scalable and maintainable the code base is. In this section, the team will present the received feedback and the changes that were made. Note that for this upload, only code that was produced by the team was uploaded. Therefore the test code ratio is quite large.

8.5.1. SIG Feedback first upload

In figure 8.1, the overall feedback is described. With a 4,2 out of 5,5-star rating, it can be concluded that the overall quality is already quite good. The most improvement can be made by looking at the duplication, unit size and unit complexity. The team was quite happy with this feedback, as it means that they were on the correct path. Especially comments, tests, and description of functions had a lot of focus in the work process because the team wanted to make sure that Bluetick would be able to use the code when the team members were gone. In this feedback, it did not say a lot about comments, so that is good.

What seems to be an issue with the feedback, is that some parts are rated as 5.5 stars, while the code is not perfect. This probably has to do with the fact that the team worked on an extension of the code that Bluetick produced. For the SIG report, the code created by Bluetick was left out. This can create the illusion that the code is very well written, while it is only a small part of the total code.

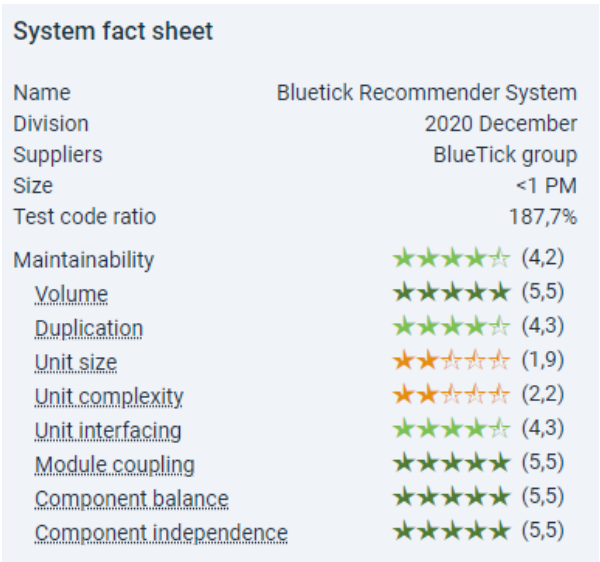


Figure 8.1: SIG report of first deadline

8.5.2. Changes made

Most of the changes were made to reduce the code duplication and unit size. The code duplication was mostly because of the tests written, which mostly relates to the test suite setup. The tests have only be

re-written slightly, as they are not the source code. For the other code duplication, most functionalities were brought into new functions. In some cases, there were only minor differences in the code. An example of this is how the partial recommender systems interact with the database. However, creating a new function while passing the connection with the required database as an argument would require about the same number of code lines in the functions. Therefore, the team choose not to do this.

After looking at a detailed version of the feedback, it can be seen that one case of code duplication that was found was located in `collection.py` from the API folder. This is were all API calls are redirected to the search API. This is indeed duplication, but removing this duplication will make the code less readable in the opinion of the team. Therefore the team chose to not focus on this particular issue. The other issues were also very hard to fix.

The unit complexity refers to the number of paths that exist in a method. This is, for example, because of if-statements in the code. Some complexity was taken away by moving parts of functions into a new function, which was the solution to some high unit size files. This fixed some complexity problems as well, but sadly many remained. Most of the files marked by the SIG system contain large functions with many switch cases. The complexity can not easily be removed from those functions, without unnecessarily splitting up the function. An example is given below, in the function where reasons are processed. Splitting up the function is unnecessary and drastically decreases the readability of the code. This function also focuses on only one process: parsing reasons into something more readable. Therefore the team did not split up these functions, and some others who have similar issues.

Another problem with the large unit size comes from the function that handles API requests. They are enormous because the input has to be divided into separate variables. The API function is responsible for calling many other functions, for processing the data from the request. These functions exist to combine all the other function, and splitting them up does not make sense. Functionality could be put in 2 different parts, but then the second function would need many parameters to take in the data from the first function properly. That did not seem helpful to the team. Some functionality could be moved, so the team decided to do what seemed like the best option for them. Sadly the remaining functions are still too large to get a good rating on the SIG platform.

```
22  /**
23   * This will process reasons into a more readable format
24   * @param tag This tag will be matched with one of the different types which are described above.
25   * @param reasons A list of reasons, that will be parsed.
26   * @returns {string|*}
27   */
28  const ReasonCreator = ({ tag, reasons }) => {
29      switch (tag) {
30          case RELEVANT_CASES:
31              return <>je <ReasonsToLink reasons={reasons}/> als relevant gemarkeerd hebt</>;
32          case HIGHLIGHT:
33              return <>je een highlight hebt geplaatst in <ReasonsToLink reasons={reasons}/></>
34          case KEYWORD:
35              return <>je op <ReasonsToItalic reasons={reasons} /> hebt gezocht</>
36          case LAWARTICLE:
37              return <>je <ReasonsToItalic reasons={reasons}/> belangrijk vindt</>
38          case ECLI:
39              return <>je op <ReasonsToLink reasons={reasons} /> hebt gezocht</>
40          case TEXT:
41              return <>je op tekst in <ReasonsToLink reasons={reasons}/> hebt gezocht</>;
42          case DOCUMENT:
43              return `het lijkt op dit document`;
44          case SEARCH:
45              return `het overeenkomt met je huidige zoekopdracht`;
46          case LARGE_TEXT:
47              return `je recent op een stuk tekst hebt gezocht`;
48          default:
49              return ``
50      }
51  }
```

Figure 8.2: The function with a too high complexity

8.5.3. SIG Feedback round 2

In this part the second round of feedback from the SIG will be discussed, the result can be seen below in image 8.3.

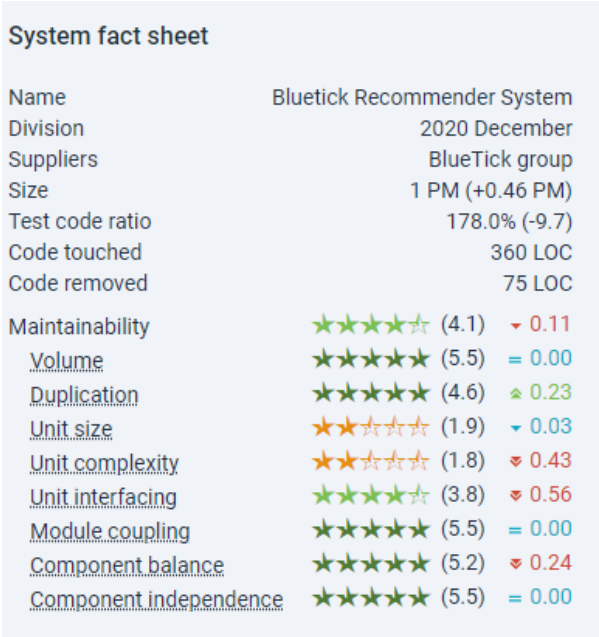


Figure 8.3: SIG report of the second deadline

The difference in scores was mostly due to a single problem: Because old code was cleaned up and removed, the functions for the API calls were now mainly written by the BEP team, instead of Bluetick's programmers. That is why the team also decided to include them in this new report, but not in the old one. These functions are, according to the metrics, poorly written functions. However, when inspecting these functions more closely, they all have a single functionality: to serve as API endpoint. They need to gather a lot of different data and combine different sources. This results in many calls, many if-statements and extended functions, sometimes with more than three parameters. The problem that the team faced was that these functions are not, in the team's opinion, more readable or more understandable when they are split up, as stated in section 8.5.2. Therefore, the team also decided not to "treat the metric" and therefore kept the code as it is now.

To end this section on a more positive note, the duplication was improved a bit. The team has actively tried to remove code duplication and were glad to see that there was now less duplication.

8.6. Team Reflection

Everybody had their own role inside the team, which mostly complemented each other. This resulted in an intuitive and healthy group dynamic right from the start. While this was the case, everybody still wanted to learn new skills and positions inside the group. Therefore, the team decided that the chairman switched with every meeting. This made sure that everyone got more experience with leading a meeting with the client, coach or others. Switching was evaluated as positive because everyone learned a lot from each other.

All team members worked in a different manner. Some worked better alone while others worked better with company. Some communicated in one way, other in another way. Nevertheless, this difference in work style was quite a positive thing, because it was in balance, just as the different interests in subjects. The one prefers working on the backend, while others prefer frontend. This made dividing tasks very easy without any conflicts.

By giving weekly individual feedback, everyone could improve on their weaker points and further develop their qualities. This weekly feedback was prepared individually beforehand to make sure all feedback could be described clearly. A list of the roles inside the group and improved points a given below:

- **Chris** was always there when others needed help. He was the rubber-duck of the group, which helped during development. Additionally, Chris was the one who really dived into the normalisation theory and gave multiple engaging presentations about this. During the project, he learned how to sound less rushed during meetings and stay focused during the long online meetings.
- **Thijs** took (almost always on the right moment) care of the entertainment during group meetings. His humour was a great addition to the meeting to keep a positive mood. He worked mostly on the backend and made a slick new version of the recommender system. In this project, he learned how important it is to review each others code and the importance of communicating well what the progress is of your feature. He also learned a lot from the rest of the team in terms of leading meetings.
- **Tijmen** is a critical thinker, to himself and to others. This characteristic became very handy as he could use this critical view to look at merge request, papers and more and give relevant feedback. This took the code and written text to the next level. In the ten weeks, he worked on all ends of the code, but his main contribution is the warm start that was implemented. Tijmen learned how to convey a solution more subtly and have an open attitude in a discussion.
- **Jeroen** communication skills were most appreciated during this project. He is more worried about the usability then whether it works or not, which complements the rest of the team. Together with his perseverance, he mostly set up new things inside the project, such as the Cypress framework and the new design for the reason of the recommendation. During the project, he learned more about when to seek help instead of figuring it out independently. Additionally, he improved upon his presenting by sometimes taking some breaks

8.7. Process Reflection

It can be concluded that the process worked quite well for the product. The meetings were well prepared and therefore, most of the time, very useful and efficient. The agile methodology worked well for the project. It made sure the team was flexible and often reflected on the direction of the project. The weekly sprints were evaluated thoroughly to improve the process's inconsistencies, and personal feedback was taken seriously, which resulted in fast adoptions in behaviour and communication. This resulted in much improvement over the weeks, on personal development and the overall process. Additional reflection is given by the Client, which is described in Appendix D

8.8. Potential improvements

While the team is quite happy with the overall process, there will always be potential improvements. The first had to do with the definition of the issues. While they were well defined, they sometimes could be split into multiple smaller issues. Some issues had a checklist of 14 points, which should have raised the question if the issue was small enough.

Additionally, some progress could have been made in keeping the `master` branch up to date with `bep-recsys`. The lack of a low amount of merge requests to `master` resulted in large merge requests with lots of features and bug fixes. This made it unclear for the client how much progress there was and which issues were already fixed or not.

Lastly, communication between the team members could have been better. Sometimes, it occurred that a bug was fixed by multiple persons. This could have been prevented by better communication of who will fix the bug.

9 Outlook

In the relatively short time span of this project, the team had to prioritize when it came to choosing what to research and implement in the final version. As a result, some recommendations and suggestions are given for the Client for further improvement.

9.1. New partial recommender systems

The new version of the recommender system makes it possible to extend easily upon the pool of techniques used to form the final recommendations. The team has some suggestions for new partial recommender systems whose introduction could have potential in the future.

9.1.1. Collaborative Filtering

As described in Section 3.4, it was predicted that applying Collaborative Filtering techniques would rarely result in significant gains to the quality of recommendations. However, if the user base of Bluetick shows signs of significant growth, it is suggested to Bluetick that they re-analyse the potential of applying these techniques.

9.1.2. Irrelevant cases

Currently, marking cases as irrelevant has little effect on the recommendations. An option to change this would be to introduce a new partial recommender system for this purpose. It could be implemented similarly to its negation, the partial system that handles relevant marked cases, with the significant difference being that it assigns negative values instead of positive ones. The easiest way to achieve this would probably be to assign the partial system a negative weight whilst internally only using positive values.

9.2. Combining Recommendations

The team foresees different approaches for Bluetick if they decide to take a step further when it comes to the process of combining partial recommender systems. Here, a few are listed.

9.2.1. Update-based influence

Currently, it does not matter for the influence of a recommender system on how much user data its recommendations are based. This raises the question of whether a system that takes this into account could possibly result in a quality gain for the recommendations. Bluetick could perform analysis on this area and introduce a weighting system that performs precisely this trick, giving more weight to systems that are based on more user data than others.

9.2.2. Normalization

A more detailed analysis could be performed on the characteristics of the score values to see which normalization process would, in theory, be most promising. However, the most straightforward and

most accurate approach would be to verify the performance of different normalization algorithms on either real user data or by introducing a labelled dataset.

It should be pointed out that it is also an option to apply different normalization algorithms simultaneously. This could be a suitable option if it is deemed desired that for some partial recommender systems, the relative distance between data points is maintained. In contrast, for others, only keeping in mind the order might result in the best recommendations.

9.3. Law Area Re-Ordering

It can be observed that some users are primarily interested in cases and jurisprudence that relate to only one or more law areas. The search interface already contains a filter for this purpose but does not seem to be used that frequently. Therefore, Bluetick mentioned their desire to asks users whether or not they want to limit their future searches to a set of law areas if these are the only ones that contain material with user interactions. The received feedback could also be used in the re-ordering of the recommender system, giving more value to cases and jurisprudence that are part of a law area the user is interested in.

9.4. Enhancing User Feedback

Recommender systems depend on the users' feedback, and the team foresee two ways of how enhancing this feedback can result in better recommendations. First of all, more precise feedback can determine the desired type and severity of the update to the recommender system. Secondly, they could be used by Bluetick to see the quality of the recommendations over time. In this section, suggestions are made where the user feedback could be improved upon in their product.

9.4.1. Explicit Feedback

As discussed in section 3.4, the current system of Bluetick allows users to mark cases and place highlights in them. Users may, however, have different reasons for performing these actions. For example, some users might put much value in the quality of recommendations and try to improve its accuracy, whereas others, might primarily use it as a 'watch-later' overview.

To better distinguish between these, users could be asked more about its motivation. Similar to how many web-shops implemented their rating system, the second phase of feedback could be introduced that asks the user, in more detail, for their motivation behind marking a case as relevant or placing the highlight. As a result, the updates on the recommendations could be or more severe, or be performed on a different partial recommender system that more suites the user's goal.

9.4.2. Implicit Feedback

In section 3.3, it was discussed that Bluetick could use implicit user data to improve their recommender system and evaluate its performance more accurately. However, due to time constraints and privacy concerns, the team prioritised the work on other areas and only started logging a few implicit feedback metrics. A more detailed analysis could be performed in this area to determine to what extent it is possible to introduce a scoring system that can accurately predict the relevance of previously encountered cases to users.

9.5. Code Quality and Maintenance

As discussed in Section 5.2, having a well-defined testing strategy is of great importance to any company in order to ease the code development process and increase maintainability.

Before working on the project, the CI/CD pipeline that ran after every commit to the Gitlab repository was limited to a check whether the docker was build or not. This could be improved by automating the tests for every merge request. This would serve as a smoke test to see if the changed code did not break anything. This would also prevent the *works on my machine* problem.

Additionally, some improvement can be made in the code style. Currently, no linter is runned in the pipeline. This will improve the consistency of the code between different developers and therefore improve the readability.

10

Conclusion

At the start of the project, the goal was to create an improved version of the recommender system, namely an optimized, faster, higher quality and more accurate recommender system than the current one. It can be concluded that creating a high quality recommender system is not trivial, especially in only ten weeks. It requires clean code, lots of data, a model that can use this data, enough time and sufficient knowledge about the subject to correctly create a working system.

As can be seen in chapter 4 of this report, not all of these requirements were met when the project started, such as the amount of data required and the extensibility of the codebase. To collect more data, implicit feedback would help, but it would not fully solve the lack of data problem. Consequently, other paths were taken to create a better basis for an improved version of the recommender system. This included a refactor of the code base, a modular system of sub recommender systems and giving control and insight to Bluetick about the recommendations' quality.

The implemented product is maintainable, extensible and traceable. The traceability is used to give the user a better feeling of the source of recommendation, contributing to a better understanding of what the underlying system is doing and how the user can contribute to its accuracy.

Additionally, some quality improvements were made to the recommender system. The user can now give more information at the start, which gives some momentum to the recommender system and improves usability. Next to this, a reordering is implemented, which is a rule-based system to further improve upon the ranking of the system's different results.

The team also kept the ethical implications of the system in mind. This is more elaborated upon in Appendix F

Lastly, there are only so many things achievable in ten weeks. This project will allow the client to efficiently improve upon the system in the near future, with some suggestions given in the outlook.

A

Info sheet

Title: A Recommender System for a Legal Search Engine
Client: Bluetick
Date presentation: January 28th, 2021

Description

In this project, an improved version of a recommender system is build for Bluetick. Bluetick is a startup that has created a new search platform for the legal sector. This platform uses optimization algorithms and AI solutions to improve the search results, and give the user a high quality and friendly tool. This search platform works together with a recommender system to recommend new cases based on search history and other user activity. The challenge of the team was to create an improved version of this system, which the team defined as a more scalable, maintainable and transparent system which would give the client the handles to increase the quality of the system.

Most research is done in the area of information retrieval and types of recommender systems. As Bluetick is still quite small, the research was mainly focused on how to recommend based on a small user collection and on how to collect more relevant user data. This research is used to further narrow down the scope of the problem and conclude were the most improvement could be made.

After the research was done, the team concluded that the most improvement could be made by restructuring the existing system in such a way that the source of recommendations was traceable and new parts of the recommendations should be easily added. This was quite a challenge to create a system that not needed a lot of user data and to create a new version of an already existing system.

But the final product is something the team is quite proud of. It is a new system that is scalable, maintainable and gives the Bluetick and the user more insight on were a recommendation was based on. Additionally, features like a warm start and reordering give already a improvement in the quality of the actual results. Lastly, the dashboard and weights give Bluetick the option to improve the system even more without any effort.

While the final product of the team is already used in production, there are always new improvements imaginable. The new system includes a normalization algorithm to combine the different partial recommendation systems, but there are more sophisticated ways to normalize. Additionally, the actual testing of the quality of the system is done manually, while this can be automated therefore this is part of the recommendations the team gives to Bluetick.

Team

Chris van der Werf
I enjoy working on innovative Data Science and AI solutions to problems we encounter in real life. During the project, I have been mainly involved on the back-end design and implementation of the new version of the Recommender System. Additionally, I contributed to the dashboard and normalization algorithm used in the system.

Thijs Nederlof
My interests are Blockchain, Cryptocurrency and Decentralised Finance, from trading bots to smart contract development. My most significant contributions to the project were done in the back-end, where I worked on creating the logic, structure, and processing of the new recommender system. I was also responsible for the code quality.

Tijmen van Graft
I am generally interested in how AI can be better integrated into our day to day live, taking over tedious/repetitive tasks from us so that we have more time for things we are truly interested in. My main contributions to this project were the warm-start, the improvement to test ability, the recommendation explanations and contributing in discussions on how structural changes should be made.

Jeroen Nelen
My personal interest mostly relates to the impact technology has on society and how to can innovate responsibly with great solutions like AI and data science. My main contributions to the project were the implementation of the weights, the front-end design and usability aspect and testing.

All team members
All team members contributed to preparing the reports and the final project presentation.

Client

Kasper Kooijman
Co-found & Developer at Bluetick

Coach

C.C.S. Liem
EEMCS, Intelligent Systems

Contact Person

Jeroen Nelen
jwnelen@gmail.com
Developer of project team

TU Delft Delft University of Technology

blue tick

B

Original Project Description

B.1. Company information

Bluetick is a start up from Amsterdam, founded in november 2019. We are developing an online search engine for the legal market. We aim to use state of the art NLP models to gain a better understanding of Dutch legal cases, and to make the life of a lawyer as easy as possible.

B.2. Project information

Legal search engines exist, but they are operating in a traditional market where little innovation has happened during the last decades. Where huge steps have been taken in the field of NLP, legal search engines have taken minor steps. At Bluetick, we use state of the art techniques to recognize which cases are similar to each other. For example: a murder case with a hammer is somewhat similar to a murder case with a baseball bat. However, with the traditional keyword-based search engines, our users wouldn't find the former case, given the latter.

For this bachelor end project the group aims to use a recommender system that finds relevant jurisprudence for the end user based on user data: different types of search queries, highlights, viewed cases, etc. The aim of this recommender system is to recommend cases the user otherwise wouldn't have found, and gain a better understanding of a particular area of law.

This project is created in cooperation with BEP group:

- Tijmen van Graft
- Thijs Nederlof
- Jeroen Nelen
- Chris van der Werf

B.3. List of questions to ask the client

- Is the topic of the bachelor project a real (i.e., not artificial or otherwise constructed) challenge faced by your organization (company or other)? Yes.
- Is it the case that there is not an off the shelf software solution that would solve the problem? There is a lot of research available on recommender systems that can be used as a starting point. However, our use case is unique and requires the necessary research.
- Is it the case that very similar solutions/products do not already exist? Legal search engines exist, but they are operating in a traditional market where little innovation has happened during the last decades. In the United States, more innovation is done on American case law, which can be used as an example for our problem, but making the translation from American case law to Dutch law is a whole new challenge.

- Is it the case that detailed specifications for the product have not already been formulated? The moments when the recommender system has to perform are specified, and the data that can be used are available. The step from data to recommendation will be up to you! (In cooperation with Bluetick, of course.)
- Does the project involve implementing a complete working prototype or a system? Yes, probably by creating an API with which our product can communicate, and by creating a database where recommendations and user behavior are stored continuously.
- Will the students have the opportunity within the scope of the project (10-11 weeks; 420 hours for each team member) to experience the whole development trajectory from problem analysis through testing? Yes
- Do you have enough capacity (i.e., time in your schedule and supporting colleagues) to hold regular meetings with the team, answer questions that arise, and guide the project to a successful conclusion? Yes, I will schedule one weekday in which I am freely available for the team. And if necessary one other meeting each week. On other days I will also be available for the team to be contacted, but I might not respond instantly.
- Do you have the time/experience to sit down with the team at the start of the project and make clear agreements about expectations and how communication will take place during the project? Four of us have a Data Science master's degree. In case development questions arise, I will make sure that our developer will also be available.
- Do the students have room to make decisions with respect to the type of solution and tools used to approach the solution? Does the project require the team to address one or more research questions? I trust four end-bachelor computer science students to be able to make solid decisions. Multiple research questions can be addressed regarding: the right database structure, combining different high dimensional vectors to find the right document similarity, combining different scoring mechanisms from different models on different scales, testing and training the model based on live user data.
- Does the project require the students to actively engage in risk management? In other words, could the project possibly potentially fail? If cases recommended by the resulting recommender system are not interesting to our users, it could be considered a failure.
- Can the project be extended if necessary (for example, if the original problem turns out to be not challenging enough)? The initial challenge would be to improve on our current recommender system: in terms of recommendations, speed, and reliability. Whilst recommendation quality can endlessly be perfected, another extension could be to introduce live learning, or to find a way to easily improve the system after gathering a few weeks of user data.

B.4. Other information

The team will preferably come to the office once a week to work there (if the situation allows). Bluetick will pay each member of the team a stipend of €200 monthly. The preferred language can either be English or Dutch. A group of around 4 students is preferred. We require the group to sign a Non-Disclosure Agreement, so we can discuss everything we want with them, and make them feel part of our company. The information shared during their final presentation or published in the thesis will not be our top secret information.

C

Project Plan

BEP - Project Plan @Bluetick

Team: Chris van der Werf
Jeroen Nelen
Tijmen Graft
Thijs Nederlof
Coach: Cynthia Liem
Client: Kasper Kooijman
Version: V1.3

Motive

Bluetick is a company that aims to make a tool for professionals working in the dutch legal industry. Their product is a smart search engine that helps the customers to speed up their research into cases. By showing them the cases they need the most, they can better interpret their current case that they are working on.

An average customer is a lawyer that works for one or more clients and on multiple cases at once and searches separately for each of his cases. He uses the results to gain a better understanding of how a judge interprets a specific law. By finding similar cases like the one he is working on, he can adjust his stance on the current case to better fit his goal. For example, he might see a case which gives insight into how he can better defend his client in his case. Bluetick's search engine tries to find the most fitting case that will help the customer the most.

Objective

Currently, the search history of the user has a limited effect on the recommendations offered by the tool. We plan on creating a search profile for the user that improves the quality of the recommendations offered to the user.

Final Product

The final product will be an improvement of a feature. The product will be a new recommender system that will recommend similar cases based on the search activity of the user within a file. Additionally, a research report will be written, which consists of a literature review, an explanation of the different options that were considered and why the implemented product needed that particular option.

Acceptance criteria

The final product is accepted if

a) There is a working version of a recommender system
b) There is an improvement in the quality (which will be based on the metrics) of the recommendations compared to the already existing baseline
c) The recommendations provided by the recommender system are verifiable and reproducible
d) The code is sufficiently maintainable by developers of Bluetick

e) The code and documentation is clean and easy to read
f) There is a research report that motivates the different design choices clearly and concise and could be used as an additional explanation for the product.

Baseline

The current system works as follows: When a user opens a new file, he will first search for relevant cases using keywords. After his first search, the recommendation system goes to work. It will select cases that it thinks will be relevant from the following data:

- Cases marked as relevant by
- Pieces of text highlighted
- Search history (either by keyword, ELCI, or part of the text)

The recommender will assign scores to cases it thinks are relevant. In a sidebar that is shown to the user, it will show a list of the cases in order of the score. By clicking on more cases and searching with different keywords/highlights/ECLI, the score is adjusted to suit the wishes of the user better.

Currently, this system is far from optimal according to the product owner. For example, there is no way to measure the quality of the recommendations, and there is no way to find similar cases in another way than the search does.

Our baseline is a recommender system that is better than the current system. Once our system works better than the old system, we will consider our project a success. To compare the two systems, we will also need a way to properly compare them in terms of recall.

Metrics

The improvements made to the system will be compared to the baseline implementation.

We evaluate the usefulness of cases to the user by looking at its behaviour:

- Whether or not the case was viewed in the first place
- Whether a case is marked as relevant
- The activity of a user on a case
- The number of highlights inserted by the user
- Whether a user interacted with a case referenced by this case

Our recommendations should be as close as possible to the optimal scenario: A list of cases that is sorted on (future) usefulness to the user.

Available data

A user can have multiple files at the same time. These files are separated from each other such that they don’t influence each other.

Currently, the following data is being tracked on a file basis:

- ECLI search history
- Keyword search history
- Highlighted text search history

- Timestamp of search actions
- A list of currently recommended cases
- Opened cases
- Saved cases
- Case found by search or by recommendation

The following information is available from every user:
- Profession in the legal system
- Years of experience
- Area of expertise in the legal system
- How much time per week a user invested in looking for other jurisprudential
- What a user spends the most time on (e.g. complex cases, simple questions, staying up-to-date)

Planning

To reach the goal of this project, we will need a schedule. In the following table are our deadlines and general activities during that time.

Week	Date	Activities	Deadlines
0	Up to 9 November	Finding a suitable project, a TU Delft Coach and getting approved	9 November
1	9 - 13 November	Getting more information from the company about the project, writing a project plan	
2	16 - 20 November	Researching, writing the research report	Research Report, Project Plan
3	23 - 27 November	Setting up the environment. Starting to work on gathering more data. Implement a more advanced logger	
4	30 Nov - 4 Dec	Start implementing content-based recommender system	
5	7 - 11 December	Deploy first recommender system/ improve logger if necessary	
6	14 - 18 December	Evaluate and improve recommender system	Hand in code for first SIG report
	21 dec - 3 jan	Holiday (Collecting data)	
7	4 - 8 January	Deploy improved recommender	

		system	
8	11 - 15 January	Writing the final paper/implement last features	Hand in code for second SIG report
9	18 - 22 January		Final report
10	25 - 29 January	Creating/finishing and giving final presentation	Presentation

Meetings

To make sure the project stays on course, and to make sure it is what the clients want, we will have a lot of contact with each other. Every week, there will be a meeting between the team and the client. These meetings will take place on Tuesday, at 9:15.

With the coach from the TU Delft, Cynthia Liem, there will be a meeting every week on Thursday 8:30.

D

Feedback from the client

Dear Chris, Jeroen, Thijs, Tijmen,

Ten weeks ago, you started working on your Bachelor Project at Bluetick. At first, you were met with disappointment, due to your expectations not being fully met. In order to build an all out recommender system, data (collection) was lacking. Despite this early setback, you decided to find a solution for our data shortage by developing a system that allows us to collect and monitor user data. Your system makes decisions based on these data, is easy to adapt, and provides our users with valuable insights.

When the definitive plan for the project was set, you could finally start the actual coding. You created your own branch and decided that it should be polished into perfection. Despite all my efforts to get it merged into master, I could not come between you and your branch. Every time I asked for it, you just started something new, which really had to be finished before it could be merged.

A week before your code was planned to be productionalized, you finally dared to hand the branch over to me. By now, the branch had become so big, that merging it into our staging environment resulted in some serious issues. However, thanks to sheer willpower and debugging skills from your side, and minor pressure from my side, you managed to fix (almost) all the problems just in time.

At this moment, your project is running in our production environment. It helps our customers to fulfill their information needs, and improves our product significantly. Thank you for your hard work. I have really enjoyed working together with you, seeing you learn and becoming part of Bluetick. I have learned a lot from you as well. Congratulations on finishing your Bachelor Project. I wish you all the best.

Kasper

E

Screenshots Recommendation Types

There are three types of recommendations. The collection recommendation page is depicted in figure E.1. Here, the recommendation is based on all activity of that particular collection. In the recent recommendations, the newest case that is relevant to a specific collection is given. This is displayed as can be seen in figure E.2. Lastly, the document recommendations are displayed as figure E.3 and are based on the current document.

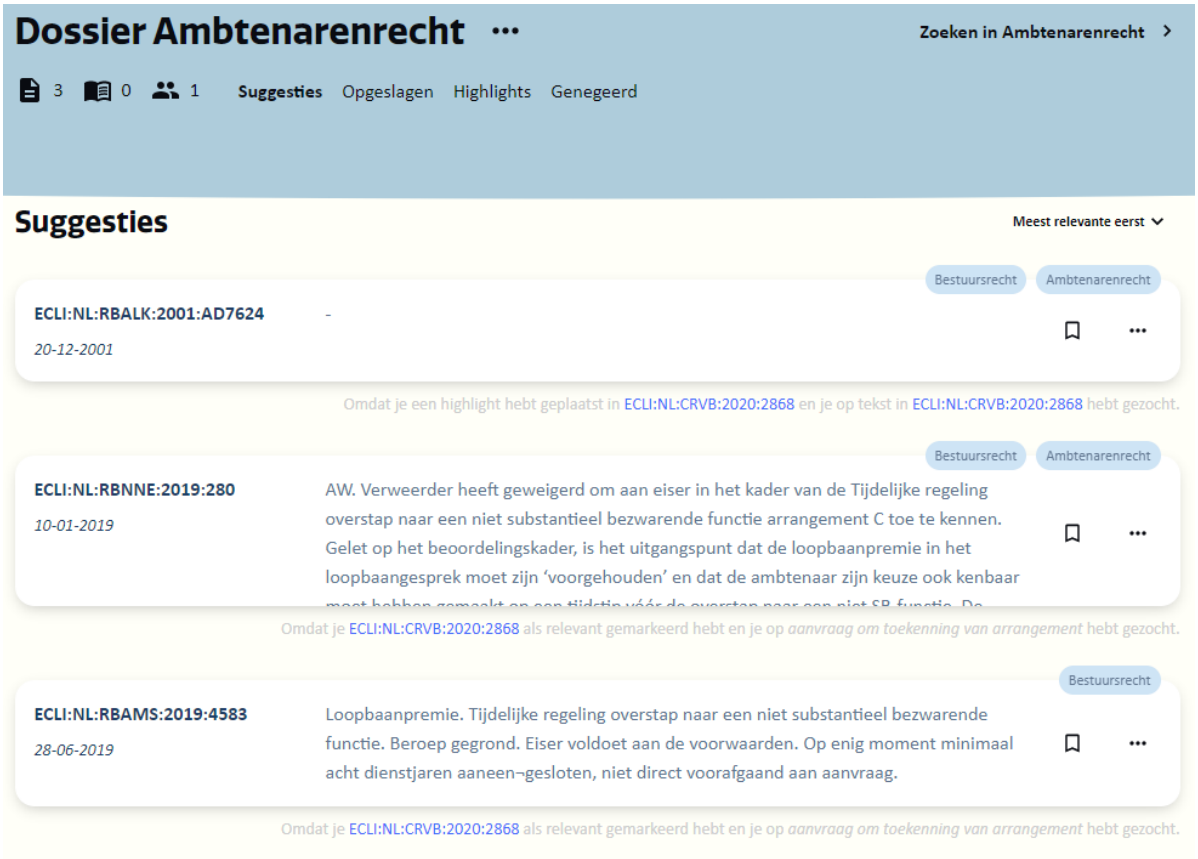


Figure E.1: Collection Recommendations

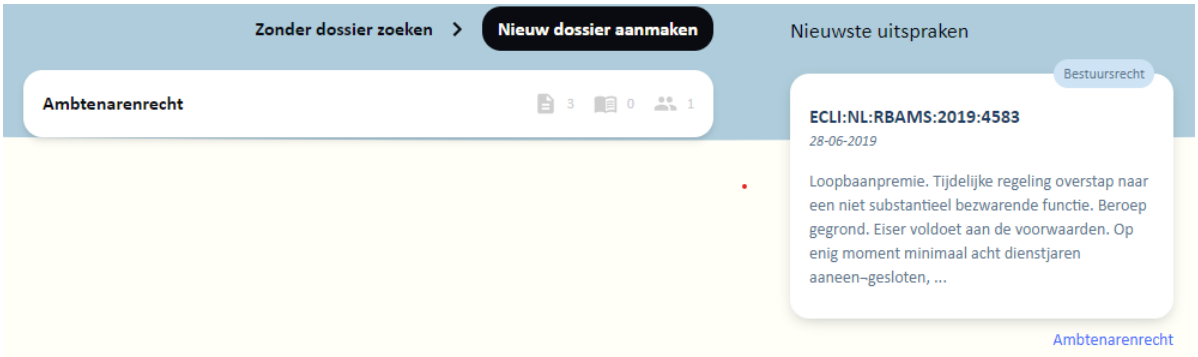


Figure E.2: Recent Recommendations

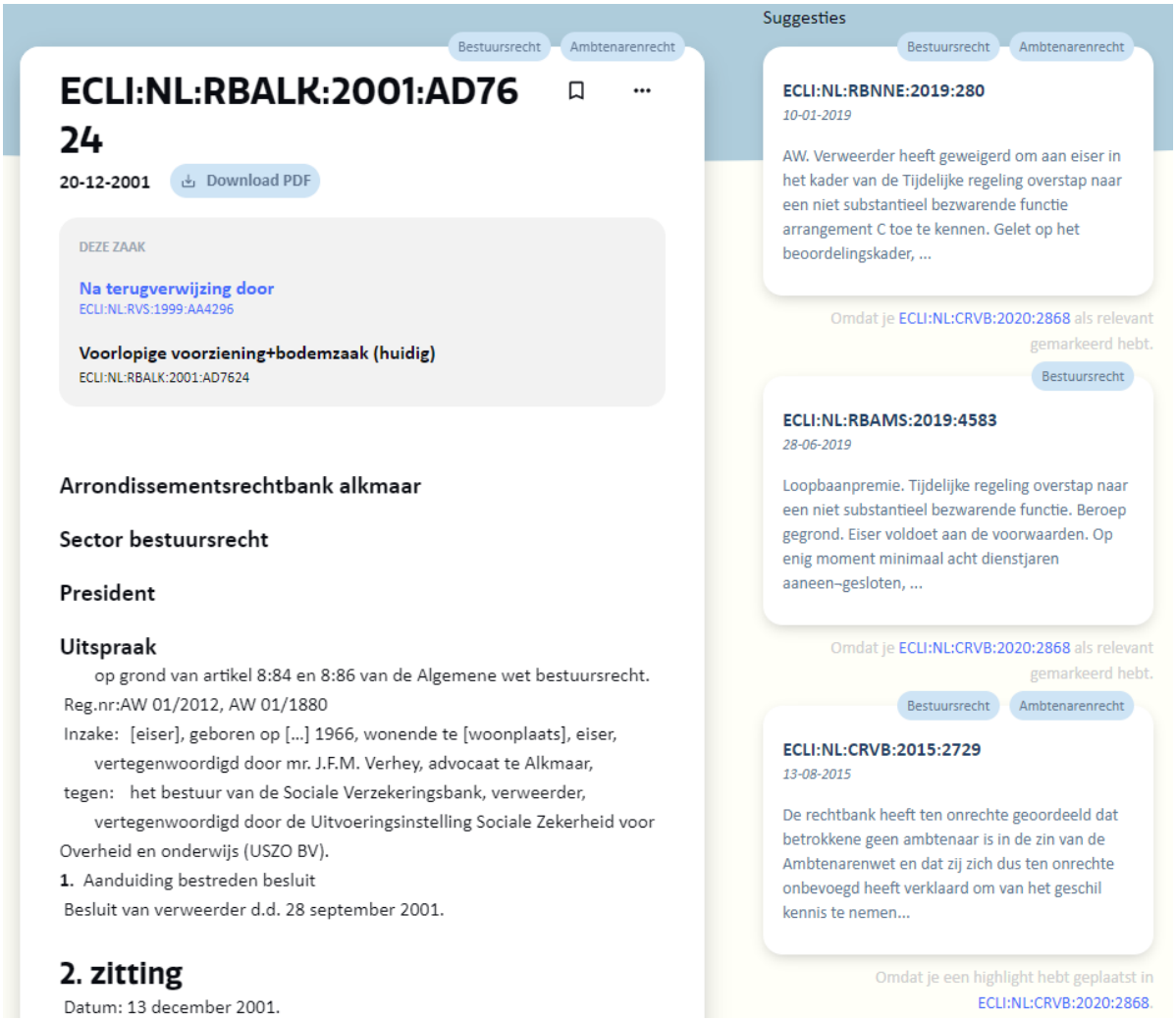


Figure E.3: Youtube Recommendations

F

Ethical implications

The ethical implications are mostly about privacy concerns. Currently, the recommender system does not use more additional data then it used to, as only the internal code has changed. However, this project added additional data collection, which is not being used at the moment, which could invade the privacy of the users.

In the first iteration of the logger, all data about user behaviour was collected: a message was sent to the back-end every few seconds if the user was active or inactive, and the scrolling behaviour was actively logged as well. After this first version was produced, the team had a talk with the person responsible for GDPR and data collection at Bluetick. He pointed out to the team that this collection was excessive, and suggested that the team changed the way the data was collected and sent to the back-end.

After re-writing the code, two major changed were introduced by the team: First of all, the activity data was longer recorded by timestamp, but it was changed to a counter that kept track how long a user was on the page (active or non-active). This was done via JavaScript, on the client's computer, to make sure nothing excessive was logged. The second adjustment was that instead of sending the data every few seconds, the data would only be sent to the back-end whenever the user moved to a new view: the clients web-browser would post an API request to a new endpoint only when he opened a new case or went back to a list view. This data contains total time active in seconds, total time passive in seconds, and scrolling behaviour. If for some reason, this information was to leak out or be misused, the implications would be a lot less severe than timestamps with which time frame a user was using his computer.

The additional data that was collected was kept to a minimum as to not collect more than necessary. A new checkbox was introduced to new users informing users about this data collection and asking them whether or not they would allow their data to be collected as well. This checkbox is not mandatory and is unchecked as default.

In the near future, this data is most likely only used to improve the quality of the recommendations and will not be sold to third parties.

The collected data is only accessible by a small number of employees of Bluetick, which creates the security of the system. The data is stored on Bluetick's back-end, so no other companies except the hosting partner can reach this data.

Bibliography

[1] Someren van Alexander et al. “Towards a Legal Recommender System”. In: *Frontiers in Artificial Intelligence and Applications* 271 (2014), pp. 168–178.

[2] Robin Burke. “Switching CF + KB: Hybrid Recommender Systems for Electronic Commerce.” In: *User Modeling and User-Adapted Interaction* 12.4 (2002), pp. 331–370.

[3] Yifan Hu, Yehuda Koren, and Chris Volinsky. “Collaborative filtering for implicit feedback datasets”. In: *2008 Eighth IEEE International Conference on Data Mining*. Ieee. 2008, pp. 263–272.

[4] Anne Boyer Julie Daher Armelle Brun. “A Review on Explanations in Recommender Systems”. In: (2017).

[5] Diane Kelly and Jaime Teevan. “Implicit feedback for inferring user preference: a bibliography”. In: *Acm Sigir Forum*. Vol. 37. 2. ACM New York, NY, USA. 2003, pp. 18–28.

[6] Gai Li and Qiang Chen. “Exploiting explicit and implicit feedback for personalized ranking”. In: *Mathematical Problems in Engineering* 2016 (2016).

[7] Edward Rolando Núñez-Valdéz et al. “Implicit feedback techniques on recommender systems applied to electronic books”. In: *Computers in Human Behavior* 28.4 (2012), pp. 1186–1193.

[8] Ladislav Peska and Peter Vojtas. “Using implicit preference relations to improve recommender systems”. In: *Journal on Data Semantics* 6.1 (2017), pp. 15–30.

[9] Peter Peska Ladislav Vojtas. “Off-line vs. on-line evaluation of recommender systems in small e-commerce”. In: *Proceedings of the 31st ACM Conference on Hypertext and Social Media, HT 2020* (2020), pp. 291–300.

[10] Ibrahim M. S. and Saidu C. I. “Recommender Systems: Algorithms, Evaluation and Limitations.” In: *Journal of Advances in Mathematics and Computer Science* 35.2 (2020), pp. 121–137.

[11] G Sasibhushana, G Vimala, and B Prabhakara. “Recommendation systems: Techniques, challenges, application, and evaluation”. In: *Advances in Intelligent Systems and Computing* 817 (2019), pp. 151–164.

[12] Software engineering group of Vienna. *Interface Pattern*. URL: `http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/interface.html`.

[13] Yaoshuang Wang, Xiaoguang Qi, and Brian Davison. “Standing on the Shoulders of Giants Ranking by Combining Multiple Sources”. In: (Jan. 2021).

[14] Ryen W White, Joemon M Jose, and Ian Ruthven. “An implicit feedback approach for interactive information retrieval”. In: *Information processing & management* 42.1 (2006), pp. 166–190.

Acronyms

API Application programmable interface. 20, 24, 34, 35

CF Collaborative Filtering. 6–8

CTR Click-Through Rate. 6, 25

DTO Data transfer object. 17, 18

ECLI European Case Law Identifier. 1, 2, 18, 19, 22, 29

EF Explicit Feedback. 8

IF Implicit Feedback. 8

IR Information Retrieval. 8

KNN K-nearest-neighbour. 6

NLP Natural Language Programming. 2

UX User Experience. 27

Glossary

case A single lawsuit which contains one file with text, the legal document, with information about where the trial took place, what year, what the final judgement was and so on. 1, 2

collection A separate collection of saved cases, search history etc. A user can have multiple collections. An example would be a lawyer who has two different customers. The lawyer will make a different collection for each client. A collection can also be called a file. 1, 19, 24

user A customer of Bluetick, someone who uses the platform/search engine . 1