FH Vorarlberg
University of Applied Sciences

Comparison of TF-IDF Model and BERT Model for the Classification of Articles in an Information Portal for a Case-Based Recommender System

Master's Thesis

Submitted in Fulfillment of the Degree

Master of Science

University of Applied Sciences Vorarlberg
Master Computer Sciences

Submitted to

DI Dr. techn. Sebastian Hegenbart

Handed in by

Anne Ilona Katharina Jovanovics, BSc
1910249008

Dornbirn, July 2021

Statement of Affirmation

I hereby declare that this thesis was in all parts exclusively prepared on my own, without using other resources than those stated. The thoughts taken directly or indirectly from external sources are properly marked as such. This thesis or parts of it were not previously submitted to another academic institution and have also not yet been published.

Dornbirn, 29 July 2021 Anne Ilona Katharina Jovanovics, BSc

Kurzreferat

In dieser Arbeit wird ein Konzept für ein Empfehlungssystem für das Informationsportal *swissmom* erarbeitet. Das Kaltstartproblem und die schwangerschaftsbedingten zeitlichen Interessensänderungen müssen in diesem Konzept berücksichtigt werden. Eine Stand der Technik Analyse über Empfehlungssysteme evaluiert geeignete Modelle für die Lösung beider Herausforderungen. Es wird eine explorative Datenanalyse durchgeführt, in der sich zeigt, dass der Schwangerschaftsmonat des Artikels ein wichtiger Indikator dafür ist, wie relevant ein Artikel für einen User ist. Da in den vorhandenen Daten die Schwangerschaftsphase der werdenden Mütter nicht bekannt ist, sind weder kollaboratives Filtern, inhaltsbasiertes Filtern, hybride Modelle noch kontextbewusste Empfehlungssysteme geeignet. Das vorgeschlagene Konzept für ein Empfehlungssystem ist daher ein fallbasiertes Modell, welches Artikel empfiehlt, die zur selben Schwangerschaftsphase gehören wie der gerade angesehene Artikel.

Dieses Empfehlungssystem benötigt für jeden Artikel den Schwangerschaftsmonat, in welchem dieser Artikel relevant ist. Diese Information ist jedoch nur bei 31% aller Artikel über Schwangerschaft bekannt. Diese Arbeit sucht deshalb einen Ansatz, welcher den Schwangerschaftsmonat anhand des Artikeltexts bestimmen kann. Die Herausforderungen bei dieser Aufgabe sind, dass nur wenige Trainingsdaten verfügbar sind und die Artikeltexte der unterschiedlichen Schwangerschaftsmonate oftmals dieselben Begriffe beinhalten, da alle Artikel vom Thema Schwangerschaft handeln. In mehreren Experimenten wird das schlagwortbasierte TF-IDF Modell mit dem kontextbasierten BERT Modell verglichen. Es zeigt sich dabei, dass der kontextbasierte Ansatz zu besseren Ergebnissen führt.

Abstract

A concept for a recommender system for the information portal *swissmom* is designed in this work. The challenges posed by the cold start problem and the pregnancy-related temporal interest changes need to be considered in the concept. A state-of-the-art research on recommender systems is conducted to evaluate suitable models for solving both challenges. The explorative data analysis shows that the article's month of pregnancy is an important indicator of how relevant an article is to a user. Neither collaborative filtering, content-based filtering, hybrid models, nor context-aware recommender systems are applicable because the user's pregnancy phase is unknown in the available data. Therefore, the proposed recommender system concept is a case-based model that recommends articles which belong to the same gestation phase as the currently viewed article.

This recommender system requires that the month of pregnancy, in which an article is relevant, is known for each article. However, this information is only available for 31% of all articles about pregnancy. Consequently, this work looks for an approach to predict the month of gestation based on the article text. The challenges with this are that only few training data are available, and the article texts of the various months of pregnancy often contain the same terms, considering all articles are about pregnancy. A keyword-based approach using the TF-IDF model is compared with a context-based approach using the BERT model. The results show that the context-based approach outperforms the keyword-based approach.

Contents

1 Introduction

Recommender systems do a pre-selection for us and present us with a manageable list of relevant options from the vast choice of available content that is available online nowadays. A recommender system helps us, for example, to choose the next movie on Netflix or to find further interesting articles to read. Netflix showed how valuable a sound recommender system is with its Netflix Prize. In 2009, Netflix held a competition in which the winning team received one million dollars (Netflix, Inc. 2021). Since 2007, ACM RecSys holds a yearly conference to discuss and present numerous new research for recommender systems. Additionally, ACM RecSys announces a new challenge every year to solve a real-world task (ACM RecSys 2021). This work proposes a concept for a recommender system for the information portal *swissmom*[1]. The following sections describe the challenges that need to be considered and the particular aims of this work.

1.1 Problem Statement

Recommender systems have to deal with different challenges that strongly depend on the data used. This work designs a concept for a recommender system for *swissmom*, which belongs to the company CH Regionalmedien AG (CH Regionalmedien AG 2021b). The website is, according to its information, "the most visited Swiss information portal with a large forum about wanting children, pregnancy, birth, baby, and child" (CH Regionalmedien AG 2021a). *swissmom* offers about 5,000 articles on the website for reading. About 285,000 users visit the website within one week, whereby approximately 90% of these users visit the website for the first time. The main challenges of a recommender system for this particular type of data are the cold start problem and changes in users' interests, which strongly depend on the course of pregnancy.

1.1.1 Cold Start Problem

A recommender system needs knowledge about users and their interests to generate sound, personalised recommendations. As mentioned before, 90% of the users who visit *swissmom's* website are on the website for the first time, which means there

[1] swissmom.ch

is no information available about these users and their interests. The problem of having no information about users is known as the cold start problem. A recommender system for *swissmom* needs to deal with this particular problem for new users. Netflix's solution to the cold start problem is to present new users with a list of movies. After the users choose the ones they like, they get access to the streaming platform. This approach allows Netflix to create a user profile for new users and generate sound suggestions. However, this option is not suitable for an information portal. Users often arrive at an article on the information portal via a Google search. If users need to rate a selection of articles before getting access to an article, most users will look for another website that provides similar information.

1.1.2 Pregnancy-Related Temporal Interest Changes

In a recommender system like Netflix, users have constant interests over a long period. It is not relevant for the user whether Netflix suggests action movie A first and action movie B five months later or vice versa. Most of *swissmom's* content is subject to a time sequence closely linked to the user. For example, a user is interested in articles about the first month of pregnancy during the first visit. It is suitable to recommend other articles from the beginning of pregnancy during this visit. However, these articles are no longer relevant for the user five months later. There are already algorithms for recommender systems that can learn temporal components, but these are either periodic temporal components independent of the user or temporal sequences of interactions (Campos, Rubio, and Cantador 2014). Periodic temporal components could be, for example, morning/evening or summer/winter. Alternatively, a recommender system can learn temporal sequences of interactions, but these do not have a fixed time interval. For example, a user first buys a printer and only then printer cartridges. However, it does not matter whether the user buys the printer cartridges a week or a month later. Therefore, it is also irrelevant whether printer cartridges are recommended to the user a week or a month later. Especially in the case of pregnancy, the time gap between interactions plays an important role.

1.2 Aim of the Work

This work aims to find a suitable approach for a recommender system that can deal with the cold start problem and the temporal interest changes. For this purpose, a state-of-the-art research is conducted, which gives an overview of current models and concepts of recommender systems. An exploratory data analysis on *swissmom's* data is carried out to investigate the behaviour of users. *swissmom's*

data consists of data that is collected with Google Analytics and data from the internally used content management system (CMS). Google Analytics tracks which articles were visited by a user and when the user visited the article. Additional information about the articles, such as the categories and weeks of pregnancy in which an article is relevant, is stored in the CMS. Based on the findings of the data analysis, the models and concepts from the state-of-the-art research will be evaluated, and applicable approaches will be selected that will help generate meaningful recommendations for users. Experiments using these methods will be carried out to determine how suitable these approaches are for creating sound recommendations. An overview of possible improvements of the recommender system concept is given in an outlook.

2 State-of-the-Art

Klahold (2009) defines a recommender system as a system consisting of a user, a context, and a set of recommended items. The context, in turn, consists of the current situation, the user profile, and the complete set of all items. The task of a recommender system is to recommend those items from the complete set that are most useful to the user under the given context. For example, when recommending restaurants to a user, the user's current location is considered the current situation and the recommended items differ depending on this location. The same goes when a user is currently viewing a comedy movie's description, and recommendations for other movies are shown beneath this description. The current situation is now the viewed movie. In this situation, recommending more comedy movies is more beneficial to the user than recommending horror movies. These two examples make it evident that the usefulness of a recommended item depends not only on the user's interests but also on the current situation.

There are four basic recommender system models. This section explains these models and the types of data used. Furthermore, context-aware recommender systems are described. This type of recommender system deals with domain-specific data and can be used with any of the four basic models. Finally, the essential evaluation metrics for recommender systems with implicit feedback data are explained.

2.1 Types of User-Item Interactions

Recommender systems distinguish between two types of user-item interactions: explicit and implicit feedback. Users give explicit feedback by rating items. The rating is usually based on a scale ranging from dislike to like. The rating scale can be, for example, an interval-based 5-star rating or a binary rating such as thumbs up/down. Unary ratings only allow the expression of liking an item. Their use is limited in explicit feedback and is used, for example, by Facebook. The only possible rating for a Facebook post is "like". Not liking a Facebook post does not automatically mean the user dislikes the post. It could be that the user either never saw the post or that the user has a neutral feeling regarding it.

Implicit feedback is gathered from the behaviour of users, for example, clicks, purchases, and shares. Implicit feedback assumes that specific user actions implicate positive feedback. Hence, implicit feedback can only use unary ratings.

While explicit ratings are often better suited to learn the interests and dislikes of users, which in turn helps generate sound recommendations, this type of feedback is more difficult to gather because it requires actions by users. There is far more implicit feedback data available than explicit ratings. These large amounts of data make implicit feedback very suitable for neural networks. Hence, research of recent years increasingly focuses on neural networks in recommender systems.

2.2 Basic Recommender System Models

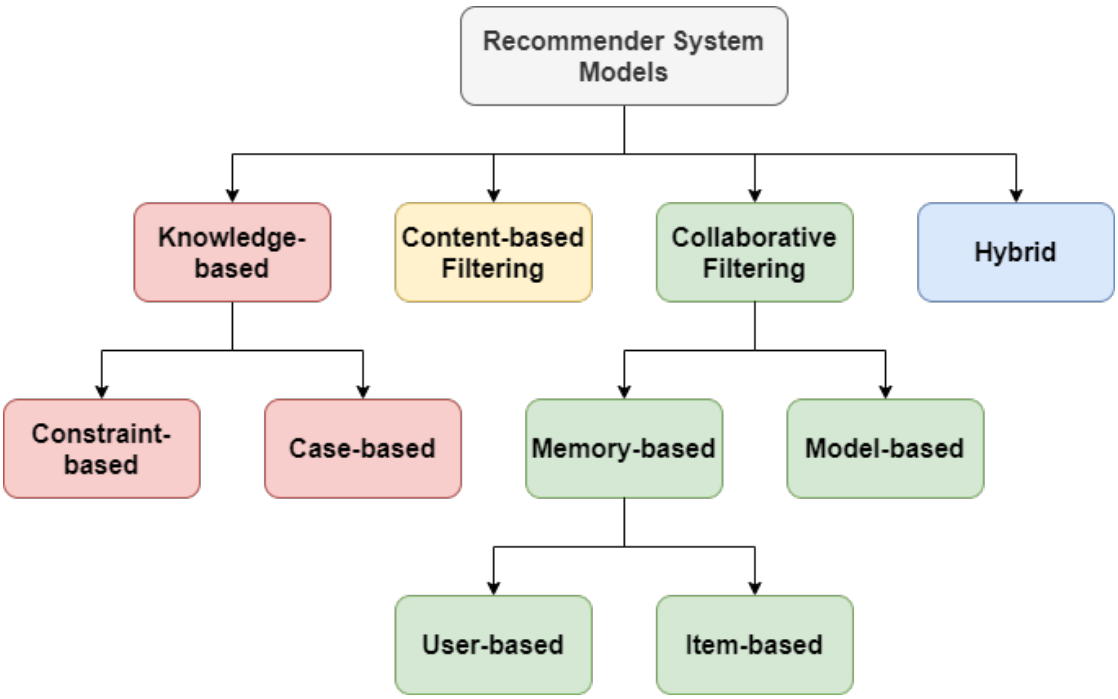

Figure 2.1: Overview of basic recommender system models

There are four basic models to design a recommender system: collaborative filtering, content-based filtering, knowledge-based models, and hybrid models (Felfernig, Jeran, et al. 2013). An overview of all models and their further categorisation is shown in figure 2.1. For the sake of completeness, it is essential to mention that

some literature considers knowledge-based models as a special case of content-based filtering.

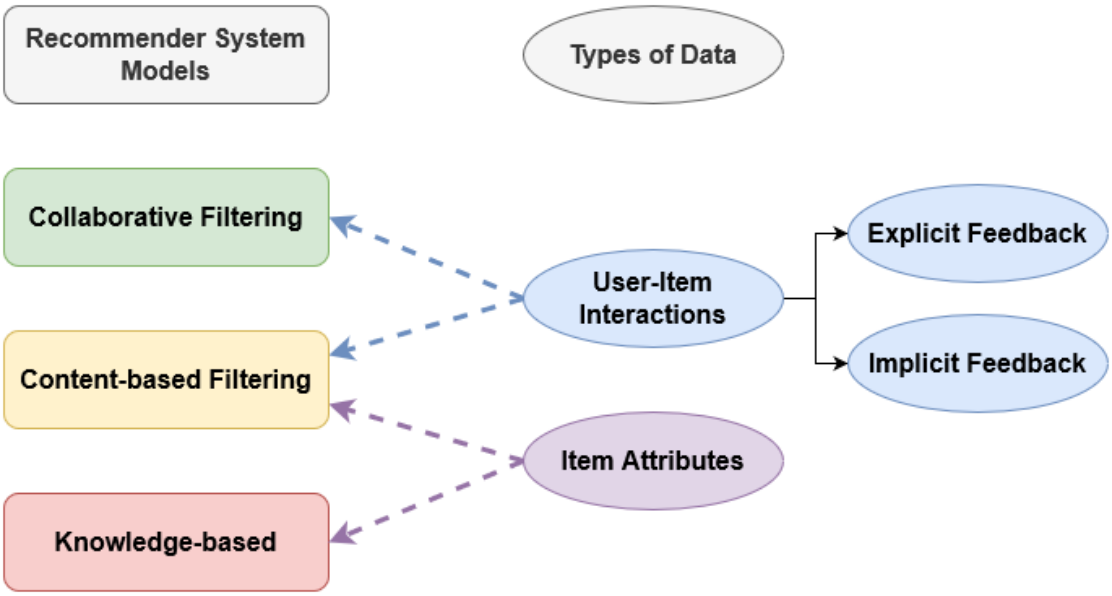

Figure 2.2: Overview of types of data and associated recommender system models

The models are mainly categorised based on the data used. There are two types of data: user-item interactions and item attributes. Figure 2.2 displays the different types of data and the associated models. Collaborative filtering models use user-item interactions. Content-based filtering models use both types of data. In contrast to collaborative filtering, content-based filtering uses only the interactions of the user for whom the recommendations are generated. Knowledge-based models only use item attributes. Hybrid models are combinations of the other three models and use the necessary data of each model.

2.2.1 Collaborative Filtering

Collaborative filtering models work with user-item interactions (Felfernig, Jeran, et al. 2013). These models predict the ratings of items by calculating similarities between users. The basic idea behind these models is that if user A liked ten movies, and user B gave positive ratings for 8 of these movies, user B would most probably enjoy the other two movies as well.

Collaborative filtering models differentiate between memory-based and model-based methods (Yang et al. 2016). Memory-based methods, also called neighbourhood-based collaborative filtering algorithms in the literature, calculate the similarity between users (user-based collaborative filtering) or items (item-based collaborative filtering) to predict unknown ratings. The idea behind user-based collaborative filtering is to find users with similar interests and predict the unknown ratings based on the k most similar users. Item-based collaborative filtering uses items that have been similarly rated by multiple users to predict missing ratings of an item. The advantages of memory-based models are their simplicity. Model-based methods use data mining and machine learning algorithms to build a model to predict ratings. Examples of model-based methods are Bayesian networks (Su and Khoshgoftaar 2006), clustering models (Ungar and Foster 1998), and latent semantic models (Hofmann 2004). Model-based methods can handle sparse data better than memory-based methods. However, model-based methods have more difficulties with users and items that are unknown during training.

One of the main problems of collaborative filtering models is sparse data. Each user interacts with or rates only a small subset of items, leading to sparse data. This makes it more difficult to find similar users. Another challenge is the popularity bias. Popular items tend to be recommended more and are therefore more visible to users, leading to more interactions between popular items and users. The opposite is true for unpopular items. Collaborative filtering models need the interaction history of a user with items to be able to find similar users. Therefore, these models cannot find similar users and recommend items for new users. The same applies to new items. These models cannot recommend items without user ratings. This is known as the cold start problem.

2.2.2 Content-Based Filtering

Content-based filtering uses attributes and descriptive characteristics of items and the interests of a user, which are learned from historical user-item interactions, to predict recommendations (Lops, Gemmis, and Semeraro 2011). Based on the items' attributes, similarities between items can be calculated. This method offers the advantage of easily explainable recommendations. Providing explanations for recommended items can increase the chance of a user interacting with the recommendation. An example of this would be "since you have read article X, you might also like article Y". In contrast to the collaborative filtering approach, content-based filtering models can also recommend new items, provided that the new items have similar properties to the present items. This model can also provide better recommendations for users with a unique taste because it does not rely on other users with similar interests. The significant disadvantage of this approach is that

the recommendations are not very diverse, and users receive only recommended items from their previous interests.

A content-based filtering system consists of three phases. The preprocessing step transforms the attributes and descriptive characteristics in a vector-space representation, whereby the representation depends highly on the type of items. The second step learns the interests of each user from the historical user-item interactions. The interests are modelled in a user profile and provide a relation between a user's interests and the extracted item characteristics from step one. The last step is predicting and ranking recommendations for a certain user based on the user's profile from step two and the item feature extractions from step one.

Feature extraction depends highly on the items used in the recommender systems. In most recommender systems, items have unstructured text attributes, for example, the description of a movie or the text of an article. The most common approach is to extract keywords from the unstructured text and represent them together with their frequency in vector-space. In order to do this, the text is broken down into individual words. Alternatively, the text can be split into phrases such as "hot dog" as well. It can be helpful to remove stop-words such as "a" or "the" to minimise the list of keywords to relevant ones. Stemming can also be used to consolidate words with the same root, for example, "bird" and "birds" or "stand" and "standing". Pazzani and Billsus (1997) recommend using only 50-300 keywords to mitigate overfitting. The frequency can be calculated by counting the occurrence of each keyword per article.

2.2.3 Knowledge-Based Recommender System

Knowledge-based recommender systems are very similar to content-based filtering. Both techniques match the interests of a user to the attributes of the items. For example, if a user on Netflix is interested in action movies, this interest can be matched to the corresponding movie genre. The difference between the two models lies in how the interests of users are collected and how long those interests are stored. Content-based filtering uses historical user-item interactions to learn the interests of each user. The learned interests are then stored in the user profile. Knowledge-based models, in contrast, do not utilise user-item interactions. Users must explicitly specify their interests before the recommender system can generate recommendations (Felfernig, Friedrich, et al. 2011). An example of a knowledge-based model would be a recommender system for houses. Users specify their needs with different filters for the price range, size, number of rooms and other attributes. The user then receives recommendations for houses that fulfil the requirements from the user's input. Depending on the item domain, the mapping of the filters

to item attributes can be complex and requires domain knowledge. An example of a complex mapping for the house recommender system would be a filter that distinguishes between "homes for singles", "small family homes", and "big family homes". This filter needs to be mapped to several attributes of a house, such as the number of bedrooms and bathrooms and the size of the house.

The interests of a user are usually not stored in a long-term user profile. Hence, if users want recommendations again at a later time, they must specify their interests again. This is particularly useful in domains where the interests of users change each time they use the recommender system. An example of this would be a recommender system for a hotel booking platform.

Knowledge-based models are divided into constraint-based (Felfernig, Friedrich, et al. 2011) and case-based (McGinty and Reilly 2011) models. Constraint-based systems use, as their name suggests, constraints to specify the requirements for the recommendations. For example, a user can select specific attribute values or a lower or upper limit. The example recommender system for houses is a constraint-based model. The system will show items that match the given constraints. Case-based systems present a list of items to a user. Each item represents a different case example, and users can choose the example that best matches their interests. For example, a case-based model for houses would present several images of houses of different sizes to the user. Based on the chosen item, the system will recommend similar items.

The main advantage of knowledge-based models is that they can deal with the cold start problem because they do not rely on historical user-item interactions to learn the interests of a user. However, this technique requires actions from the user before the system can generate recommendations, which is not suitable for all applications. Therefore, knowledge-based systems are mostly used in complex item domains. Content-based filtering systems and case-based systems use similarity functions to generate recommendations. Hence, both systems have the problem that the recommendations are hardly diverse. If users do not like one recommended item, they usually like none of the recommendations. A straightforward approach to mitigate this problem is the bounded random selection strategy. Instead of the top k recommendations, k items are randomly selected from a list of top bk items (Smyth and McClave 2001).

2.2.4 Hybrid Recommender System

Each of the discussed models has different advantages and disadvantages. A hybrid recommender system (Burke 2002) is a combination of different models, for example, a collaborative filtering model combined with a knowledge-based model. This

combination can lead to better results in specific scenarios as it allows leveraging the different advantages of the models.

2.3 Context-Aware Recommender System

Additional information such as location or time can be substantial to create sound recommendations depending on the domain. For example, when recommending restaurants, the location where the user wants to eat is a decisive criterion for the recommendations. Contextual information always depends on the situation in which the recommendation is made. For *swissmom*, the user's current pregnancy phase at the time when the recommendations are to be made is decisive for generating recommendations. Recommender systems that work with context data have many names in the literature and are called context-aware, context-based or context-sensitive recommender systems.

Temporal data is frequently used as additional context. This special case of a context-aware recommender system is called a time-sensitive recommender system (Campos, Rubio, and Cantador 2014). Time can have two different roles in a recommender system: sequential or contextual. Temporal sequences can be used to adapt to changes in the interests of a user. One way to do this is by penalising older ratings with a decaying factor. Another possibility with time sequences is to learn temporal sequences of interactions. For example, users who bought furniture for a baby will generally purchase diapers in the future. A time-sensitive model can learn less obvious sequences of interactions.

Time as a context is used to recommend different content based on the time. For example, an online shop for clothes will recommend different types of clothes based on the current season. Time here does not mean the actual date, but for example, morning, summer, or weekend. The categorisation of the time depends on the domain of the data. For an online clothing shop, categorising time into months or seasons makes more sense than a detailed division into morning and evening or days of the week. For Netflix, on the other hand, such a precise division is applicable because consumption patterns for films and series during the week are different from those at the weekend. In a time-sensitive recommender system, each user-item interaction additionally contains the time of the interaction. The time can be processed either during pre-filtering, training or post-filtering. In pre-filtering, all ratings with an irrelevant time are filtered out. For example, if the recommender system wants to create recommendations for the summer season, all ratings without the time category summer are removed before training. In post-filtering, training is done with all ratings. The recommendations are generated, and all recommendations with the wrong time category are filtered out. Lastly,

with a more complex model, the time category can be included in the training process. In this case, the model is trained to ignore all irrelevant time categories. Combinations are also possible. For example, the time is handled during training as well as post-filtering.

2.4 Evaluation of Recommender System

The performance of a recommender system is determined through offline evaluation (Herlocker et al. 2004). Historical user-item interactions are used to calculate metrics in offline evaluation, which makes the evaluation results comparable between different algorithms. However, the significance of the results is limited. For example, a recommender system could generate the perfect recommendations for a user. However, if this user has never interacted with the generated recommendations in the historical data, the evaluation result for this user will be poor. The final performance of a recommender system can, therefore, only be evaluated in an online setting. This means that users will receive recommendations from the finished recommender system. It is then measured how often users interact with the recommended items. Since this work designs the concept for the recommender system but does not implement a ready-to-use prototype, only metrics for offline evaluation are described in this section. Furthermore, these metrics are limited to those that can be applied to implicit ratings, as the data of *swissmom* consists of implicit feedback.

2.4.1 Accuracy

The goals of a recommender system, and thus what the evaluation measures, can be different. For example, it may be important that the recommendations are as diverse as possible. In most cases, accuracy is defined as the goal (Shani and Gunawardana 2011). Accuracy indicates how often the recommender system correctly predicts if a user likes or dislikes a particular item. Accuracy can be measured well and is therefore suitable for evaluation. Basically, for implicit feedback data, accuracy is measured by generating recommendations for each user and comparing how many of these recommended items are in the test data. The following metrics are all based on measuring accuracy.

2.4.2 Metrics

A recommender system predicts the rating a user would give for all user-item combinations with a missing rating. As implicit feedback has no ratings, the rating is 1 if the user has interacted with this item and 0 otherwise. One way to

evaluate the accuracy is to calculate the deviation between the actual rating and the calculated rating. This approach is especially suitable for explicit feedback data but not for implicit feedback. Ranking evaluation methods such as utility-based ranking evaluation and receiver operating characteristic (ROC) (Fawcett 2004) are suitable for evaluating recommender systems with implicit feedback.

Utility-Based Ranking Evaluation

Utility-based methods measure the usefulness of an item to a user. Average reciprocal hit-rate (ARHR) (Deshpande and Karypis 2004) is an often-used metric for implicit feedback data. Equation 2.1 shows the calculation of ARHR for user u. Equation 2.2 shows the calculation of the global ARHR value, which is the mean ARHR of all users.

$$\mathrm{ARHR}(u) = \sum_{j \in I_u, v_j \leq L} \frac{1}{v_j} \tag{2.1}$$

$$\mathrm{ARHR} = \frac{1}{m} \sum_{u=1}^{m} \mathrm{ARHR}(u) \tag{2.2}$$

The total number of users is denoted as m, and I_u is the set of all items of user u that are in the test data. For each user u a list of recommendations is generated. The items in each list are ranked from 1 to L, where 1 represents the highest relevance for a user. The rank of item j in the generated recommendation list is denoted as v_j. The hit-rate is calculated for each generated recommendation that exists in I_u. ARHR is the sum of all hit-rates for one user. For example, items A and B exist in the test data for user Z. The generated recommendation list of length $L = 2$ contains item B at rank 1 and item C at rank 2. The hit-rate for item B is $1/1 = 1$ and for item C 0 because this item does not occur in the test data. ARHR for user Z is $1 + 0 = 1$. The higher an interacted item is ranked in the predicted recommendation list, the higher its hit-rate. The highest hit-rate, achieved by an item at rank 1, is 1. The hit-rate of a generated recommendation that is not present in the test data is 0.

Receiver Operating Characteristic (ROC)

ROC methods measure recall and precision. Recall is the proportion of interacted items that the recommender system recommends. The proportion of recommended items with which the user interacted is called precision. The interacted items are the items in the test data set. This set of items is also called ground truth and is

denoted as G in the following equations. The calculation of precision and recall for a user are shown in equations 2.3 and 2.4.

$$\text{Precision}(t) = 100\frac{|S(t) \cap G|}{|S(t)|} \tag{2.3}$$

$$\text{Recall}(t) = 100\frac{|S(t) \cap G|}{|G|} \tag{2.4}$$

The higher the metrics, the better the recommendations generated. However, a high precision usually means a low recall and vice-versa. The length of the list of recommendations is denoted as t, and $S(t)$ is the set of top t recommended items. For example, the items A, B, and C are in the test data, which means $G = \{A, B, C\}$. The recommended list contains the items A and D, therefore, $t = 2$ and $S(t) = \{A, D\}$. The calculation of precision and recall results in $100 * 1/2 = 50\%$ and $100 * 1/3 = 33\%$, respectively. One of the two recommendations occurs in the test data, leading to a precision of 50%. Recall is lower than precision because out of the three items in the test data, only one item is actually recommended. The two metrics, precision and recall, are calculated for different t and displayed in a graph as a Precision-Recall curve for interpretability.

The ROC curve is easier to interpret. This curve also needs two metrics to be calculated: recall referred to as true-positive rate (TPR) and false-positive rate (FPR). The equations of TPR and FPR for a user are given in 2.5 and 2.6, respectively. The set of all items is denoted as U. TPR should be as high as possible, while FPR should be as low as possible.

$$\text{TPR}(t) = 100\frac{|S(t) \cap G|}{|G|} \tag{2.5}$$

$$\text{FPR}(t) = 100\frac{|S(t) \setminus G|}{|U \setminus G|} \tag{2.6}$$

3 Data Analysis

This section describes the conducted explorative data analysis on *swissmom's* data. The findings will be used to evaluate which recommender system models are suitable.

The data set consists of data collected via Google Analytics from 7 February until 7 March 2021 and contains 2.5 million entries. Google Analytics tracks the interactions of users with the website. Each interaction consists of a unique user-id, the article's URL, the article's id, and the date of the page view. Additional information about the articles was retrieved from *swissmom's* CMS. Articles that are relevant in specific weeks of pregnancy have the week number as an additional property. Generally, each article can be assigned to multiple weeks with a value between 0-1, determining the article's relevancy. For simplicity, each article has the week with the highest relevancy assigned. Unfortunately, the user's week of gestation is unknown. This section assumes that the user is in the same week of pregnancy as the first article visited by this user, which has an assigned week of pregnancy. This assumption is based on the fact that most users visit an article from swissmom because they found a link to the article during a Google search. Therefore, it stands to reason that users are looking for information about their current week of pregnancy and visit an article relevant to the user's current phase of pregnancy. This additional information makes it possible to analyse if a relationship between the pregnancy week of articles and users exists.

3.1 Categories of Articles

swissmom has a total of 4,530 articles about the topics "wanting children", "pregnancy", "baby", "child", "family", and other subjects. Figure 3.1 shows the number of interactions per category in per cent. In addition, the number of articles per category is also shown in per cent. About 45% of all visited articles are on pregnancy, making this the most popular topic among users. About a third of all articles are written about pregnancy, which makes "pregnancy" the topic with the most articles. The second favourite topic of users is "baby" with 36% of all interactions. The remaining three topics each account for less than 6% of all interactions. This makes sense for the topic "wanting children" as less than 5% of

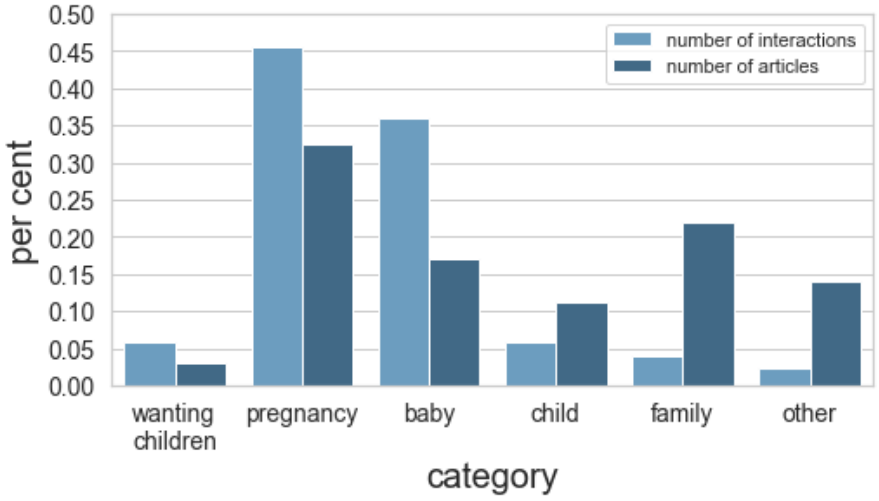

Figure 3.1: Number of interactions and articles in per cent per category

all articles are written about this subject. However, even though about 20% of all articles cover the topic "family", only 4% of all interactions are in this category.

It was further analysed how many different categories were visited by each user during one month. Since the goal of the recommender system is to encourage users to read more articles, it is more interesting to analyse the behaviour of users that viewed multiple articles during the analysed period. Hence, the number of different categories was counted for users with at least five interactions. The median number of categories for these users is two. 25% of all users viewed articles from only a single category, and 75% of all users viewed articles from two or fewer categories, which suggests that users are generally interested in articles about one or two topics.

3.2 User

Figure 3.2 shows a boxenplot with the number of interactions per user on the x-axis. Most users visit very few articles.The median is two interactions per user, and 75% of all users visited three articles or less between 7 February 2021 and 7 March 2021. The data set contains 849,105 unique users, of which 48% have only viewed exactly one article. A few users visited more than 200 pages in the period under review.

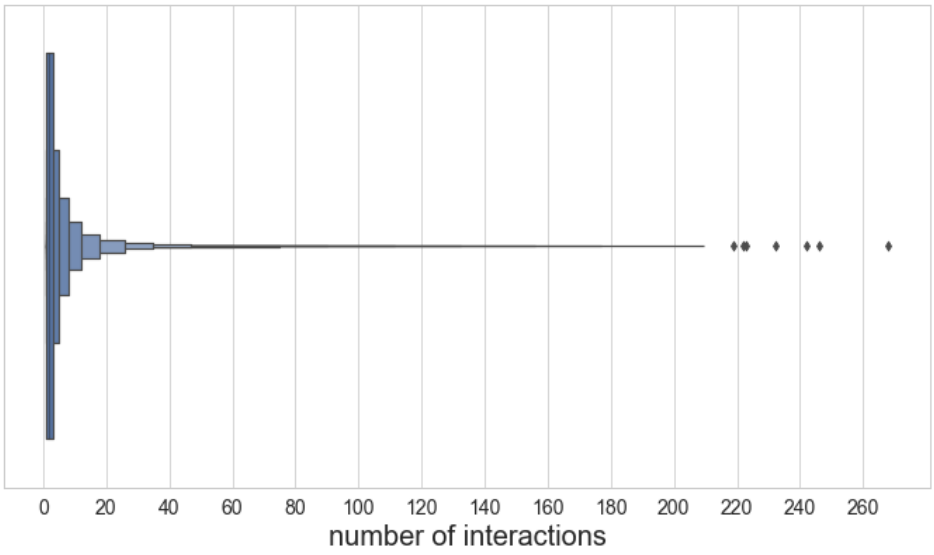

Figure 3.2: Number of interactions per user

3.3 Article

Figure 3.3 shows a boxenplot with the number of interactions per article on the x-axis. The data set contains 4,141 articles. Only a few articles are very popular

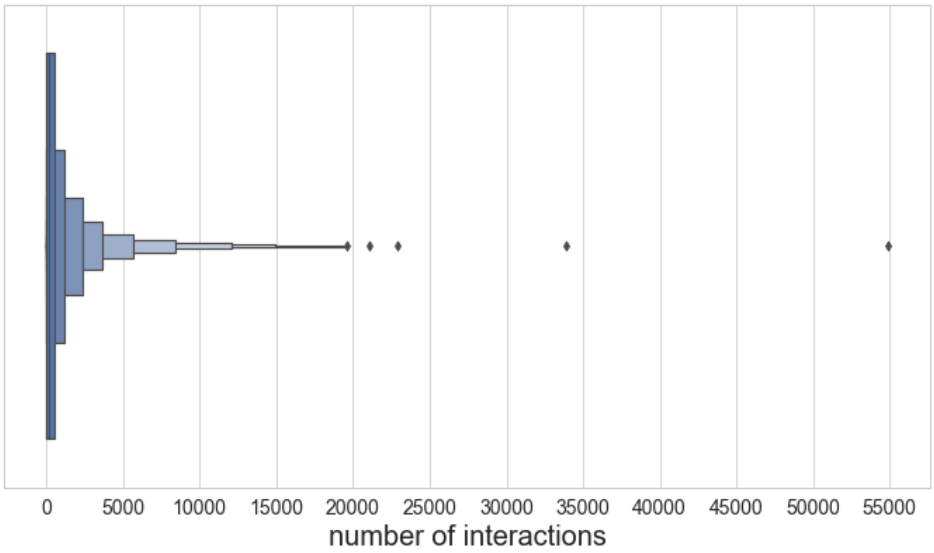

Figure 3.3: Number of interactions per article

and visited more than 10,000 times. 75% of the articles are visited 517 times or less during one month. The plot clearly shows a popularity bias in the data. The two articles with the most visits have over 30,000 and over 50,000 views, respectively. These articles have a disproportionate number of interactions compared to the others. Both pages are ordinary articles and not the landing page, an error page or something similar.

3.4 Date

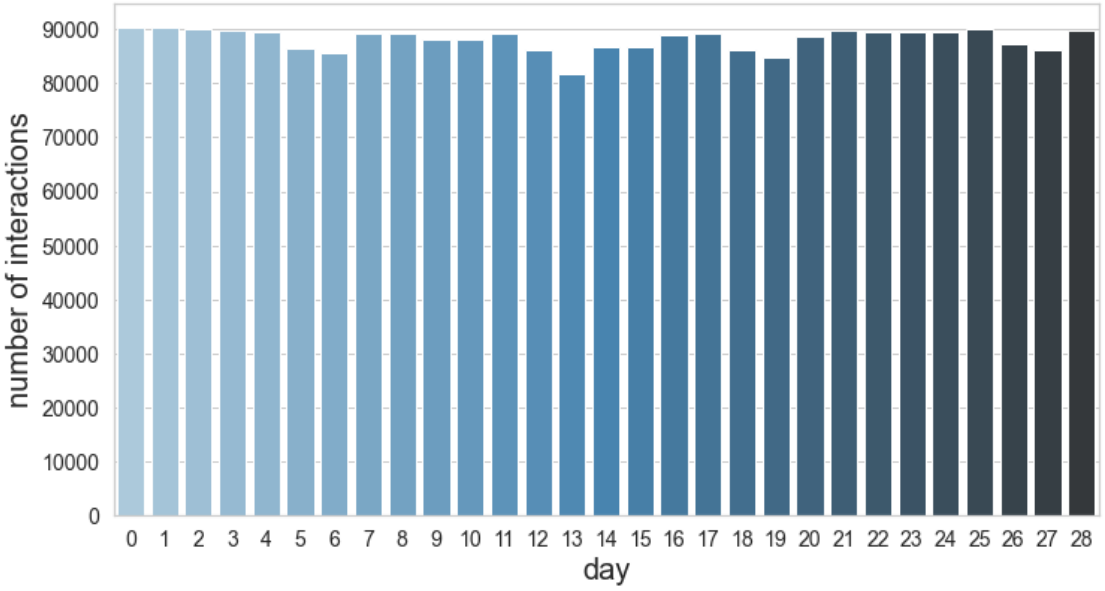

Figure 3.4: Number of interactions per day

Figure 3.4 shows a barplot with the number of interactions per day on the y-axis. The x-axis shows the number of the day, ranging from 7 February 2021 (day 0) to 7 March 2021 (day 28). The interactions are fairly evenly distributed.

3.5 Articles with a Week of Pregnancy

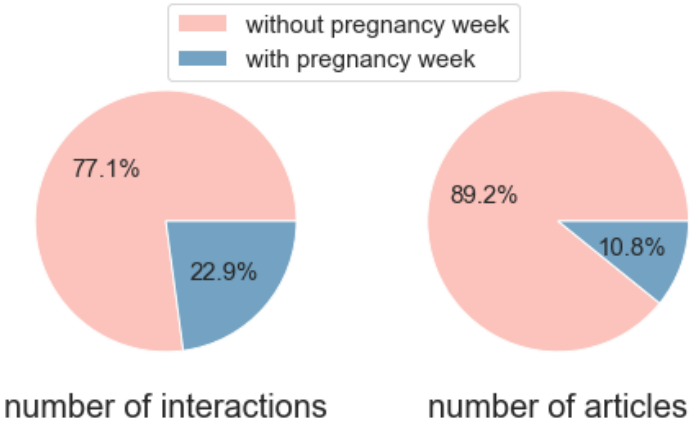

Figure 3.5: Per cent of interactions with week of pregnancy and per cent of articles with week of pregnancy

In this section, the analysis of articles with an assigned week of gestation is described. Almost all of these articles belong to the category "pregnancy". *swissmom* offers articles for weeks 5-42. Figure 3.5 shows that 22.9% of all interactions were

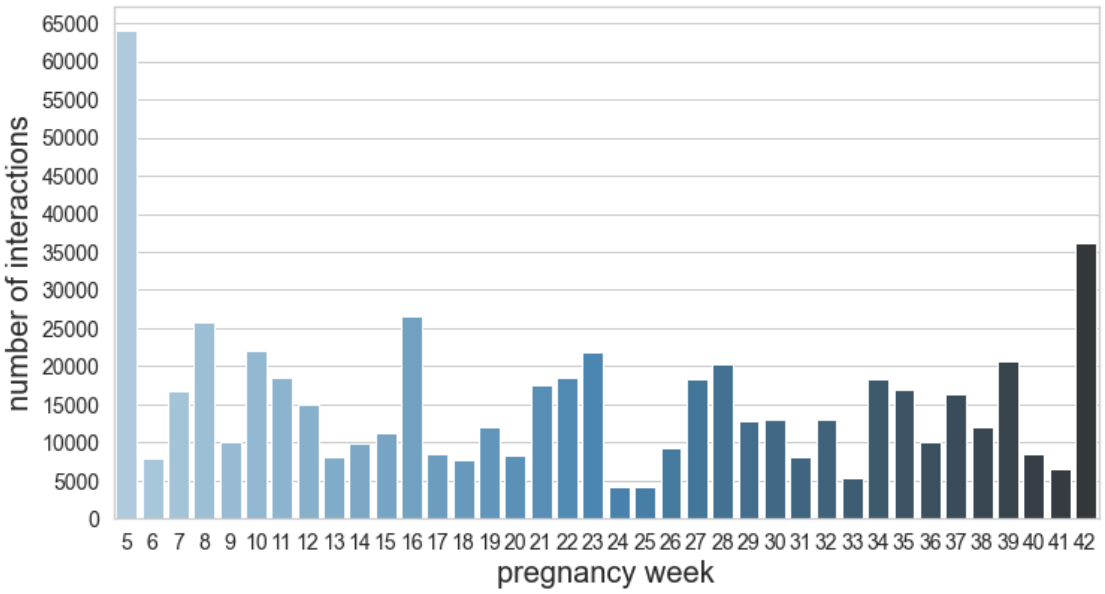

Figure 3.6: Number of interactions per week of pregnancy

with an article with an assigned week of pregnancy, and 10.8% of all articles have a week of pregnancy assigned.

Figure 3.6 shows a barplot with the number of interactions per week of pregnancy on the y-axis and the week of pregnancy on the x-axis. Articles relevant in week five were visited over 60,000 times. The interactions per week of gestation are unevenly distributed, with most interactions in week five followed by week 42 with over 35,000 interactions.

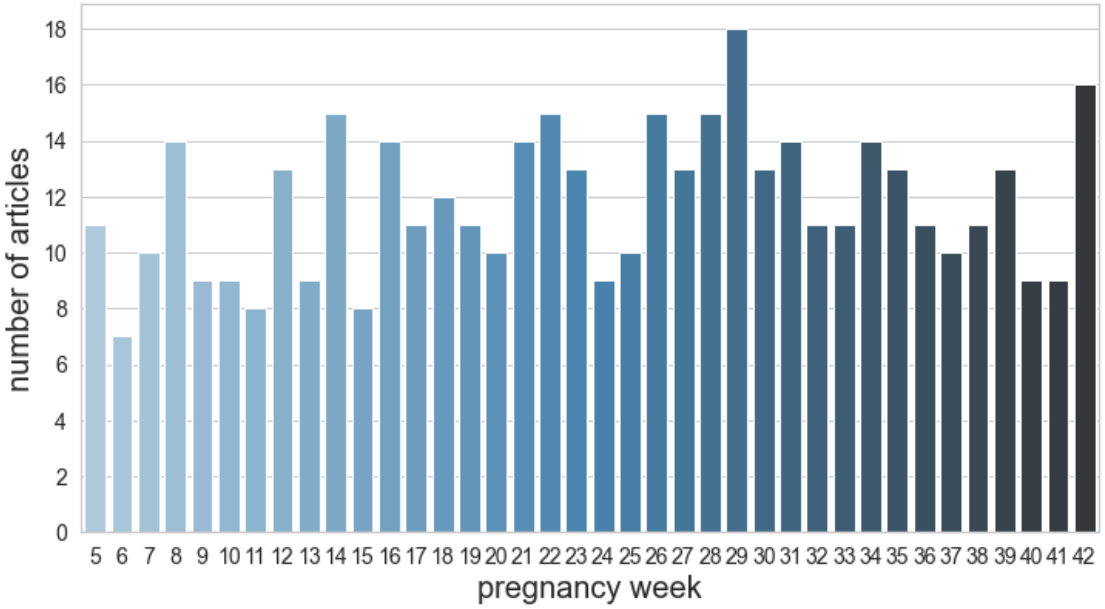

Figure 3.7: Number of articles per week of pregnancy

To better understand the distribution of interactions per week of gestation, the number of articles per week is analysed. A barplot visualising this is displayed in figure 3.7. The y-axis shows the number of articles, and the x-axis shows the week of pregnancy. Each week has at least seven and at most 18 articles. The distribution of articles per week differs from the distribution of interactions per week, meaning that the high interest of users in week five cannot be explained by a greater variety of articles in this week. Users seem to be most interested in articles about the beginning (week five) and the end (week 42) of the pregnancy.

3.6 Span of Relevant Weeks of Pregnancy for Users

This section analyses how large the span between the lowest and highest week of pregnancy is that a user viewed. The span is calculated for each day individually. For example, if user A visited two articles on day one, and one is relevant in week five, and the other is about week six, then this counts as a span of one week. If the same user A visits the website seven days later and reads three articles, all about week 12, then this counts as a span of zero weeks. The span is calculated daily and not over the entire month. Otherwise, the span is generally longer for those users who read articles on different days than those who visited the website only once. In addition, this analysis better shows the short-term interests of users during a single visit. Furthermore, the span is calculated for all users who viewed at least two articles with an assigned week in one day. Calculating the span for users who only viewed one article would always result in a span of zero weeks and distort the result. A span of zero weeks means that a user only viewed articles from the same week of pregnancy. A span of two means that a user has viewed articles where the weeks of pregnancy are two weeks apart. This would be the case if the user viewed articles from weeks 35, 37 and optionally 36. The data is split into three categories: users with two, three and more than three interactions per day.

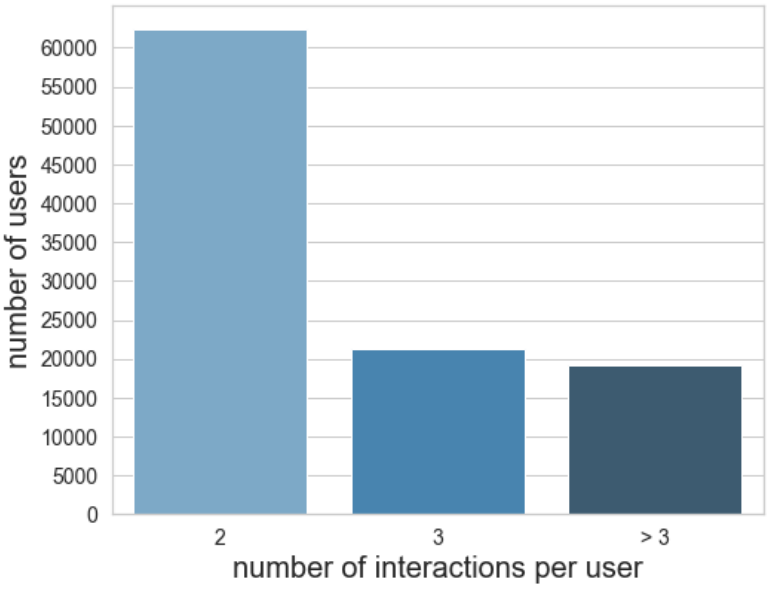

Figure 3.8: Number of interactions per user and date

Figure 3.8 shows that over 60,000 users looked at exactly two articles on a single day. About 20,000 users viewed exactly three articles, and slightly fewer users

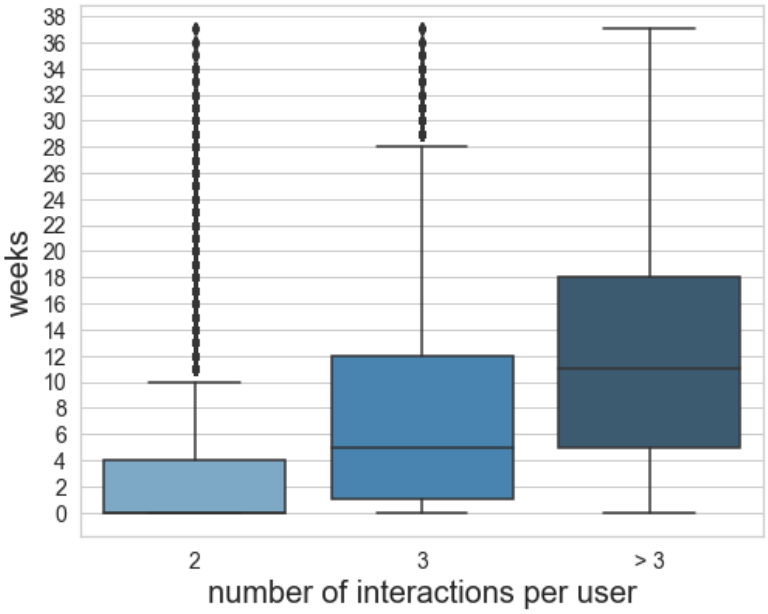

Figure 3.9: Span of relevant weeks of pregnancy per user and date

visited more than three articles. Figure 3.9 shows the span between weeks of pregnancy per user and day as a boxplot for each of the three categories. The y-axis shows the span in weeks, and the x-axis shows the number of interactions per user. The median span increases with an increasing number of interactions. The median span is zero weeks for users with two interactions per day. For users with three and more than three interactions per day, the median range is five and 11 weeks, respectively. This analysis shows that users are interested in information from a more extended period and not only from a specific pregnancy week.

3.7 Week of Pregnancy

The user's week of pregnancy is unknown in the data set. To analyse if there is a correlation between the week of pregnancy of an article and the week of pregnancy of a user, it is assumed that the user is in the same week of gestation as the first article viewed by this user, that has an assigned week.

Figure 3.10 shows the known week of pregnancy of articles on the y-axis and the assumed week of users on the x-axis. The interactions are counted for each combination of the users' and articles' week of gestation. For example, the bottom left cell value of the plot contains how many articles about week five were viewed by

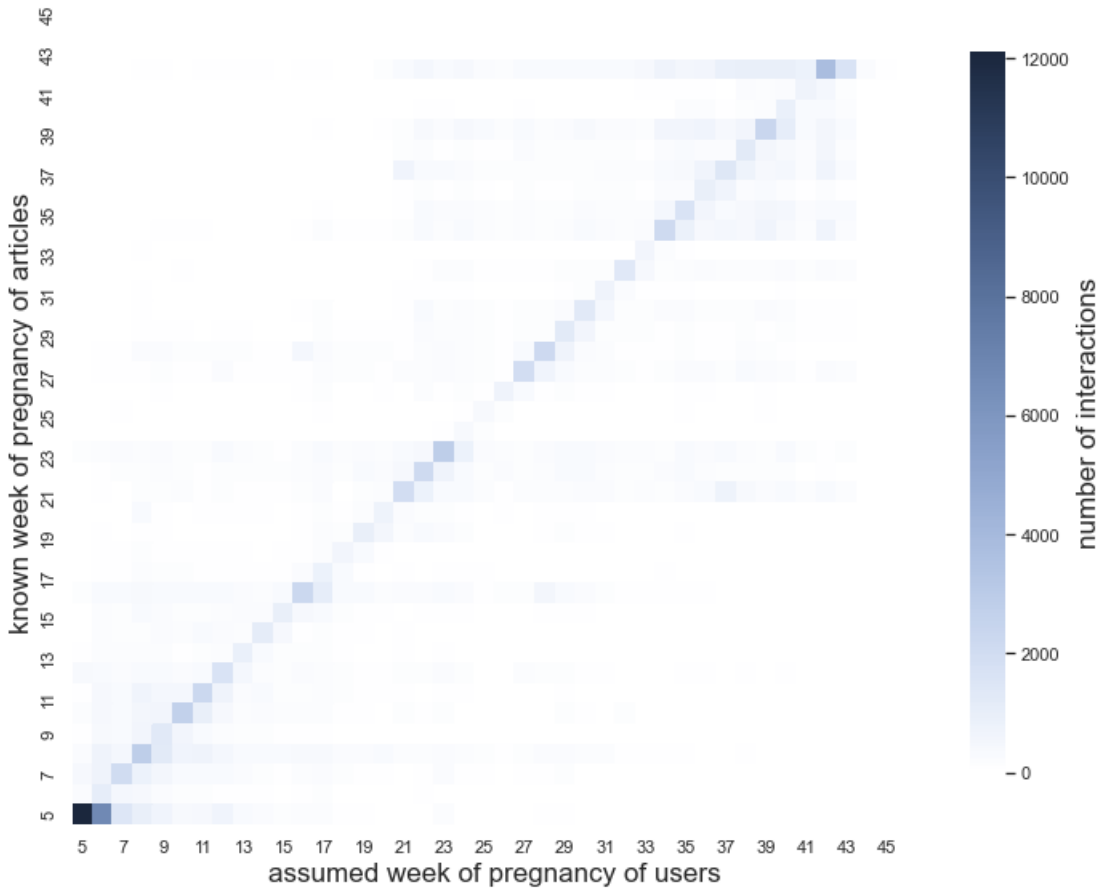

Figure 3.10: Significance of the known week of pregnancy of articles for the assumed week of pregnancy of users

users in the assumed fifth week of pregnancy. The values are visualised by colours, with high values represented by dark colours and low values by light colours. Most articles were viewed by users in the assumed fifth week of pregnancy. These users mainly looked at articles about the same week, showing that articles about a particular week are more significant to users that are hypothetically in the same week. This is also true, albeit to a lesser extent, for most other weeks. Articles about a particular week of pregnancy are visited more by users who are not just in the same week as the article, but in the same phase of pregnancy, whereby the phase can span over several weeks. This can be seen well in figure 3.10, as darker colours, and thus higher values, occur not just on the diagonal but also close to it. For example, users in the hypothetical week eight of pregnancy visited articles about the eighth week the most. This can be seen in figure 3.10 because

the darkest blue of the column with the label "8" is in the row of the eighth week of pregnancy of articles. Lighter blue shades can be seen in rows 5-7 and 9-11, indicating that these weeks are also significant to users in the eighth week. The correlation between the week of pregnancy of articles and users is 0.68, which is a moderate positive correlation.

3.8 Conclusion

The analysis showed that we have to deal with the cold start problem since 75% of all users visit three or fewer articles, which means there is not much information about them, and a user profile cannot be created. Hence, collaborative and content-based filtering techniques are poorly suited to generate recommendations for these users because both models rely on historical user-item interactions to create user profiles. Three or fewer historical interactions per user are not enough to extract interests for content-based filtering or to find similarities to other users for collaborative filtering. The popularity bias in articles also suggests that collaborative filtering is inapplicable as this technique is known to have problems recommending unpopular items.

Analysing the categories of articles has shown that most users view articles from a maximum of two categories. From these findings, it can be deduced that a recommender system should generally recommend articles from the same category, which can be achieved with content-based filtering or a knowledge-based model. The approach with a knowledge-based model has the additional advantage that more complex rules can be introduced. For example, suppose it turns out that users who view articles from the category "family" generally also visit articles about children. In this case, a knowledge-based model could recommend further articles about the topics "family" and "child" to a user who is currently reading an article about family.

Examining the categories of articles also suggests that "pregnancy" is the most significant topic because articles from this category are visited most frequently by users. Therefore, sound recommendations should be generated for users that are interested in this subject. Furthermore, the analysis of the week of pregnancy implies that the pregnancy phase of a user is essential, and articles about the current pregnancy phase of the user are more interesting. To solve the challenge with pregnancy-related temporal interest changes, collaborative and content-based filtering techniques can only be used in a context-aware recommender system where the pregnancy phase of users is known and used as an additional context. In this case, techniques such as pre-filtering or post-filtering could generate a list of

recommendations containing only articles from relevant weeks of gestation. However, the user's pregnancy phase is unknown. Therefore, neither context-aware recommender systems nor collaborative and content-based filtering models are suitable. The data analysis showed that users mainly visit articles belonging to the same phase of pregnancy. Instead of using the user's week of gestation, which is unknown, the article's week of gestation can be considered an indicator of the pregnancy phase of a user. Hence, the proposed recommender system concept is a case-based model. For all articles about pregnancy, this model uses the currently viewed article and generates recommendations based on the pregnancy week of the article. For all other categories, the category of the article indicates the interest of a user and articles from the same category will be recommended.

4 Methodology

The data analysis indicates that the assumed week of pregnancy of a user correlates with the week of pregnancy of the visited article. For the recommendations to be meaningful to a pregnant user, articles about the pregnancy phase of the user should be recommended. The data analysis has shown that articles from a span of 11 weeks are especially interesting to users. The data from *swissmom* does not contain any information about the pregnancy phase of the user. Therefore, this work does not attempt to learn this because no evaluation is possible. *swissmom* editors determined the most relevant week of pregnancy for 481 articles. These are 31% of all articles from the category "pregnancy". Thus, for 69% of articles about pregnancy, the week still needs to be determined to recommend relevant articles to a pregnant user. In this section, different methods are applied to the articles to determine the pregnancy phase of an article. Due to the small amount of data, the article's month of pregnancy is determined instead of the week. There are different ways to divide pregnancy into months. In the approach chosen in this work, a month consists of four weeks. Thus, at 42 weeks, a pregnancy lasts eleven months instead of the known nine months.

When working with text data, two representations of data are common: bag-of-words representation (Joachims 1998) and sequential representation. The bag-of-words representation uses the frequency of words in a text. This approach focuses on keywords and loses the order of words in the text and, therefore, the context. The sequential representation maintains the order of words in a text. Machine learning models can thus better understand the context of words. An experiment for each type of text representation is conducted to determine the articles' month of pregnancy. The first experiment uses the bag-of-words representation, and the second experiment the sequential representation. Thus, it can be investigated whether keywords or context are better suited to determine the article's month of gestation.

A commonly used approach for bag-of-words representation is the term frequency - inverse document frequency (TF-IDF) (Salton and Buckley 1988) model, used together with various text preprocessing methods and regression models in the first experiment. The state-of-the-art approach that uses contextual information in texts is the bidirectional encoder representations from transformers (BERT) model

developed by Devlin et al. (2019). This model is used in the second experiment.

As mentioned before, both experiments will predict the article's month of gestation. In machine learning terms, the predicted attribute is called label. There are two types of prediction tasks: classification and regression. Classification is used for categorical label types, such as "Cat" and "Bird", and regression is used for numerical label types. Simply put, the output of a classifier is one of the given labels. This predicted label is compared to the actual label and is either the same or not. The result of a regressor is a number. In this case, it is possible to calculate the distance between the actual label and the predicted label. Larger deviations between the predicted label and the actual label thus lead to a larger error. The months are numerical and larger deviations between the prediction and the actual month of pregnancy should lead to a larger error in the evaluation. Therefore, regression is used to determine the month of gestation.

Both approaches will use 5-fold cross-validation for training and evaluation. In k-fold cross-validation, the training data are split into k equal-sized data sets, called folds. $k-1$ folds are used for training. The last fold is used for evaluation. This step is repeated k times, with each fold being used exactly once for evaluation. The average of all results of the k evaluations is then the final evaluation result. This method is used for small data sets where not enough data is available for distinct training and evaluation sets.

Root mean squared error (RMSE) is used as a metric for the performance of the regression models and is given in equation 4.1. The vector containing the actual labels is denoted as $\mathbf{y}$, and $\hat{\mathbf{y}}$ contains the predicted labels. The number of records in the data set for which labels were predicted is denoted as n.

$$\mathrm{RMSE}(\mathbf{y}, \hat{\mathbf{y}}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2} \tag{4.1}$$

4.1 TF-IDF Model

In this section, the article texts are represented using the TF-IDF model. This model uses the bag-of-words representation, where the normalised frequency of words is calculated with TF-IDF. This type of representation considers the individual words of the texts separately, but it pays no attention to context or the order in which the words occur. The preprocessing of texts is an important step that can significantly influence the performance of the subsequently used regression algorithm. Several preprocessing methods are described in this section. Additionally, the experiments using the TF-IDF model will be explained.

4.1.1 Text Preprocessing Methods

First, some text mining terminology is defined. Texts are usually called documents. The set of all documents is called corpus. Texts are split into smaller letter sequences, usually on a word basis. These letter sequences are called tokens. The document “the ball and the shoe” is broken down into the tokens “the”, “ball”, “and”, “the”, “shoe”. This example shows that two tokens can consist of the same word. Words are usually referred to as terms. Terms are unique and have a frequency. So in the example text, there is only one term “the” with the frequency of two but two tokens “the”.

Bag-Of-Words Representation

Text data must first be transformed from an unstructured representation into a structured representation before it can be used in machine learning. The TF-IDF model uses the bag-of-words representation, which represents each document as a vector. The dimension of the vector corresponds to the number of terms in the corpus. Each dimension of the vector is assigned to exactly one term. The vector itself contains the frequency of the term in the respective document. Figure 4.1 illustrates an example of a bag-of-words representation for a corpus consisting of the two documents “blue, blue, blue is a colour”, and “the ball is blue”. The bag-of-words representation of each document is shown as a row vector. The corpus contains the terms “blue”, “is”, “a”, “colour”, “the”, “ball”. The frequency of

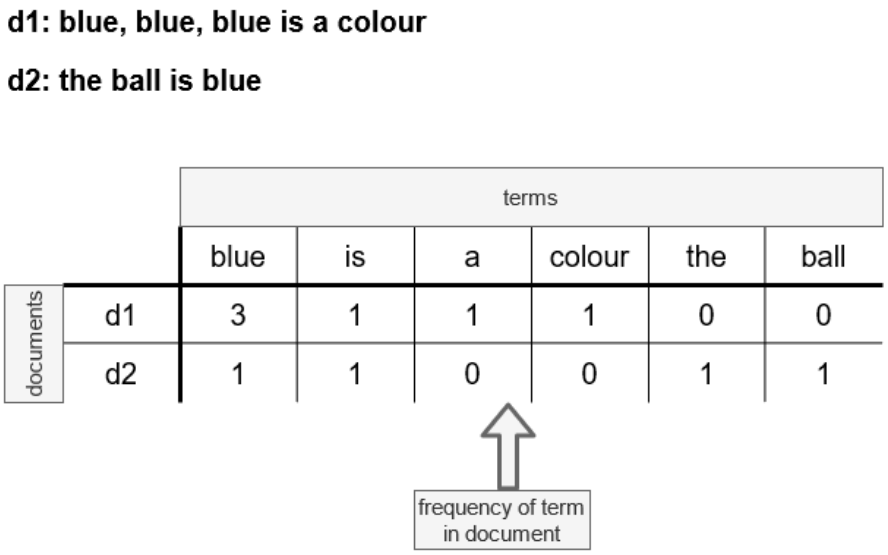

Figure 4.1: Example of bag-of-words representation

each term within the document is counted and stored in the vector. For example, "blue" occurs three times in document d1 and one time in document d2.

Alternatively, a binary representation is possible, which uses 1 instead of the frequency if a term is present in the document and 0 if the term does not occur.

Text Extraction

All non-textual elements are removed during text extraction, which is necessary, for example, when using web page content. The text on web pages is embedded in HTML tags. These tags must be removed so that only the plain text is processed. Furthermore, punctuation marks such as comma and points also need to be removed. After all documents of the corpus consist only of plain text, each document is split into tokens.

Stop-Word Removal

An essential step of preprocessing is the handling of stop-words, which are common words such as "the" or "of" that frequently appear in texts but have no special meaning. Stop-words depend on the language of the text, and lists of stop-words exist for several languages. Stop-word removal is a common approach that reduces the number of terms in the corpus (Saif, Fernandez, and Alani 2014).

Stemming & Lemmatisation

Stemming and lemmatisation are algorithms to consolidate words with the same root (Porter 1980). For example, the words "tree" and "trees" have the root "tree" or the words "run" and "running" have the root "run". In most scenarios, it does not make sense to distinguish between words with the same root. For example, a user interested in articles containing the word "tree" most likely wants to read articles about "trees". Using stemming or lemmatisation can reduce the total number of terms. If this data is then used to train a machine learning algorithm, the reduced number of features can reduce overfitting and improve accuracy.

Stemming reduces each word to its stem. The stem does not necessarily correspond to a word. For example, the words "argue" and "arguing" are reduced to the stem "argu". There are various stemming techniques, such as lookup tables and suffix stripping. In lookup tables, the unstemmed word variants are stored for each stem. Each word is then looked up in the table during stemming, and the correct stem is determined. In suffix stripping, various rules are applied to reduce words to the stem. For example, the endings "ly", "ed", and "ing" are removed from each word. Lemmatisation reduces each word to its lemma. Unlike a stem, a lemma is still a

valid word. The lemma of "argue" and "arguing" is "argue". Lemmatisation uses more complex rules that involve the context of words. This technique is, therefore, more complex and slower but more accurate. For example, the words "universal" and "university" are reduced to "univers" when stemming. Lemmatisation, on the other hand, recognises that these words have different meanings. The lemmas are "universal" and "university".

TF-IDF

The less frequent a term appears in documents, the more information can be gained from this term. This can be taken into account when the occurrence of each term is normalised with a frequency-based normalisation. With this approach, terms that occur in many documents are down weighted. TF-IDF is an often-used frequency-based normalisation. The term frequency (TF) is multiplied with the inverse document frequency (IDF). TF for the term i and for document d is calculated by counting the occurrence of term i in document d. Equation 4.2 shows the calculation of IDF for the term i.

$$\mathrm{IDF}_i = log(n/n_i) \tag{4.2}$$

The number of documents in the corpus is denoted as n, and n_i is the number of documents containing the term i. IDF_i is 0 for a term that is present in every document.

d1: green, green, and blue

d2: the ball is blue

term	blue	green	
TF for d1	1	2	term frequency for term i and document d1
n	2	2	total number of documents
n_i	2	1	number of documents containing term i
IDF	0	0.7	log(n / n_i)
TF-IDF	0	1.4	

Figure 4.2: Example of TF-IDF calculation

Figure 4.2 shows an example of the calculation of TF-IDF for a corpus containing two documents. TF-IDF is calculated for the term "blue" and "green" for document $d1$. "blue" appears in all documents which always results in a TF-IDF of 0. "green" occurs twice in the first document, resulting in a TF-IDF of 1.4.

4.1.2 Experiments

Each article from *swissmom* has several attributes that contain text. These include title, headline, label, teaser title, description, text, and categories. Attributes such as the title or description summarise the article's content and contain words that are particularly relevant to the article. If the frequency of words is counted in all textual attributes and not only in the text, more weight can be given to these words. Therefore, all textual attributes of the article are used for the representation of an article.

The data is preprocessed in different ways to evaluate which method or combination of methods is most suitable. The following methods are used:

1. No Preprocessing
2. Stop-Word Removal
3. Stemming
4. Stop-Word Removal combined with Stemming
5. Lemmatisation
6. Stop-Word Removal combined with Lemmatisation

The Python platform Natural Language Toolkit (NLTK)[1] is very popular for text preprocessing. NLTK's German stop-word list and implementation of the Snowball Stemmer were used. NLTK, unfortunately, does not offer an implementation for lemmatisation in German, so the Python library spaCy[2] was used for this task. spaCy is a very helpful library for building text preprocessing pipelines in natural language processing (NLP) applications.

[1] www.nltk.org/
[2] spacy.io/

After preprocessing, the normalised term frequency is calculated using TF-IDF by utilising the implementation of scikit-learn[3]. The month of pregnancy is used as the label. The label was linearly scaled from 2-11 to 0-1 with min-max normalisation. Several regression algorithms were used for training to rule out the possibility that the results were poor because an inappropriate regression model was chosen. Ridge regression (Hoerl and Kennard 1970) and Support Vector Regression (SVR) (Drucker et al. 1996) are trained and evaluated using 5-fold cross-validation. The implementations of scikit-learn were used for training and evaluation.

Pazzani and Billsus (1997) recommend using only 50-300 keywords to mitigate overfitting. Without restriction, the number of features is between 13,000-19,000, depending on the preprocessing method. Therefore, 300 was used as the lower limit for the number of features, and the upper limit is all features. In addition, the regression models are trained with 3,000 and 10,000 features to evaluate the best number of features. In each case, the k most frequent terms of the entire corpus are used.

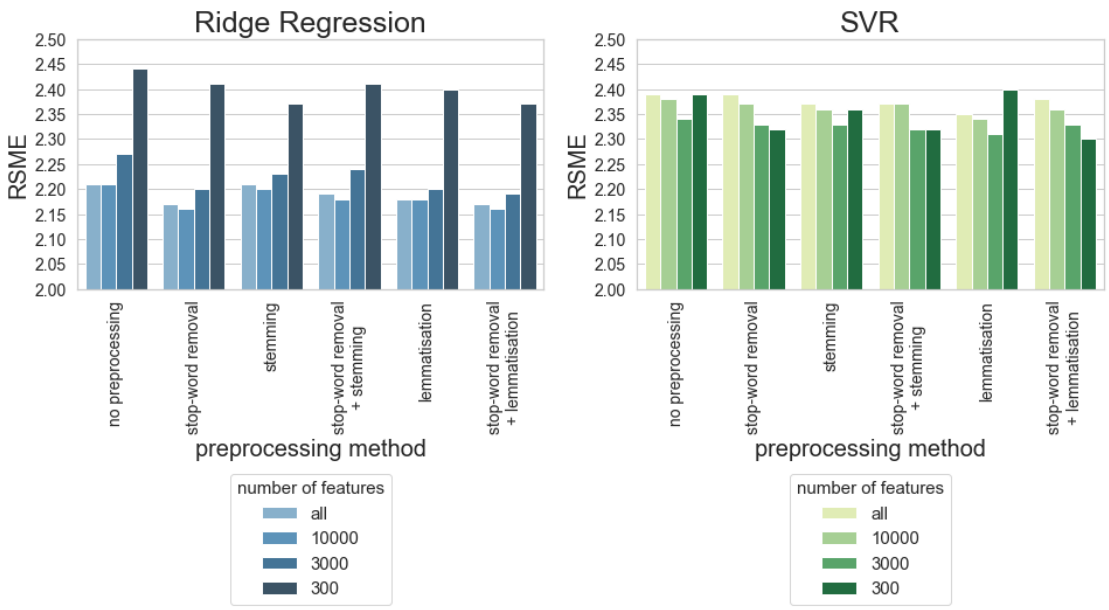

Figure 4.3: RMSE for TF-IDF experiments

[3]scikit-learn.org

Results

Table 4.1 presents the RMSE in months for the different preprocessing methods, number of features and regression models. These results are also displayed visually as barplots in figure 4.3. A separate plot was made for each regressor. The bars are grouped by preprocessing method within each plot. Within a group, the results for the different number of features are shown. The y-axis shows the RMSE and is cut off below 2.0 for better visualisation of the results.

	number of features			
	all	10,000	3,000	300
no preprocessing				
ridge regression	**2.21**	**2.21**	2.27	2.44
SVR	2.39	2.38	2.34	2.39
stop-word removal				
ridge regression	2.17	**2.16**	2.20	2.41
SVR	2.39	2.37	2.33	2.32
stemming				
ridge regression	2.21	**2.20**	2.23	2.37
SVR	2.37	2.36	2.33	2.36
stop-word removal + stemming				
ridge regression	2.19	**2.18**	2.24	2.41
SVR	2.37	2.37	2.32	2.32
lemmatisation				
ridge regression	**2.18**	**2.18**	2.20	2.40
SVR	2.35	2.34	2.31	2.40
stop-word removal + lemmatisation				
ridge regression	2.17	**2.16**	2.19	2.37
SVR	2.38	2.36	2.33	2.30

Table 4.1: RMSE for TF-IDF experiments

Ridge Regression: The worst results with an RMSE over 2.35 are achieved with 300 features independent of the applied preprocessing method. All preprocessing methods perform identical or slightly better when using 10,000 features instead of all. Slightly worse results are achieved with 3,000 features. The best results are achieved with 10,000 features.

SVR: SVR generally performs better with decreasing number of features, which is inverse to how ridge regression behaves. Comparing the results achieved with

300 features and 3,000 features for preprocessing methods without a stop-word removal step, the performance of SVR deteriorates. Hence, the ideal number of features is 300 if a stop-word removal step is applied and 3,000 otherwise.

Ridge regression performs worse than SVR when using only 300 features. However, with more features, ridge regression outperforms SVR. The best RMSE for SVR is 2.30 and for ridge regression 2.16, which means ridge regression is better suited than SVR to predict the month of pregnancy.

The worst results with ridge regression were obtained when no preprocessing is applied, closely followed by stemming. The results for stemming can be improved if stop-word removal is also applied during preprocessing. However, better results are achieved when using only stop-word removal. This means stemming deteriorates the performance in this case. Lemmatisation outperforms stemming. However, better results can be achieved if stop-word removal is applied before lemmatisation during preprocessing. The best results are achieved with 10,000 features and either only stop-word removal or stop-word removal and lemmatisation combined, which means lemmatisation neither deteriorates nor improves the performance in this case. Hence, ridge regression with 10,000 features and stop-word removal as the preprocessing method should be used to predict the month of gestation. Nevertheless, the reader should notice that the absolute differences between the results are very small with a maximum difference of 0.28, and therefore one method is not better than another with absolute certainty.

4.2 BERT Model

This section describes the experiments conducted with the BERT model. BERT is a pre-trained language representation model that uses a transformer architecture. Transformer architectures, in turn, are based on the concept of self-attention. This section will shortly explain the mechanisms behind self-attention and transformers and give an overview of BERT and how it was trained. The design of the experiments is also described in this section, and the results are discussed.

4.2.1 Self-Attention

Attention and self-attention mechanisms were initially invented to overcome challenges in sequence-to-sequence tasks such as translating a text. Self-attention receives N inputs and calculates attention weights for each input. This can best be explained with a short example. The example sentence is “My name is Alice.”. Let us assume the word “is” is masked, and we want to predict the correct word. In order to do this, we look at the other words of the sentence. We will give the

Figure 4.4: Visualised attention weights for an example sentence

most attention to the word "name", and, based on the singular form of this word, we can assume that the missing word is "is". If self-attention receives the example sentence as input, it will calculate attention-weights for each word of the sentence. The weights for the input word "is" are visualised in figure 4.4 as blue blocks. The dark blue colour means that the word "name" is very significant to the word "is". The visualisation was created with exBERT[4].

There are different attention functions for calculating the attention weights, with additive attention (Bahdanau, K. Cho, and Bengio 2015) and dot-product attention (Luong, Pham, and Manning 2015) being the most popular methods. BERT uses scaled dot-product attention, which will be explained in this section. Self-attention receives a sequence of dimension N as input (Karim, Raimi 2019). For example, the sequence "Hello World" has dimension two and consists of `input 0 = Hello` and `input 1 = World`. In the first step, each input i is transformed into a word embedding. Word embeddings are numeric representations and consist of a vocabulary, where each word in the vocabulary is associated with a numeric vector (Mikolov, Le, and Sutskever 2013). The special feature of word embeddings is that contextually similar words are assigned to similar vectors. Word embeddings are usually learned using machine learning.

Before we can calculate the attention weights, the matrices query Q, key K, and value V need to be calculated from the input. Figure 4.5 visualises the matrix operations for calculating these matrices. The rows of the input embedding matrix X consists of the word embedding vectors. This matrix is multiplied with the weight matrix W^Q which results in matrix Q. The same is done with weight matrices W^K and W^V to calculate the matrices K and V, respectively. The three equations are given in 4.3, 4.4, and 4.5.

$$Q = XW^Q \tag{4.3}$$

[4]https://huggingface.co/exbert

$$K = XW^K \tag{4.4}$$

$$V = XW^V \tag{4.5}$$

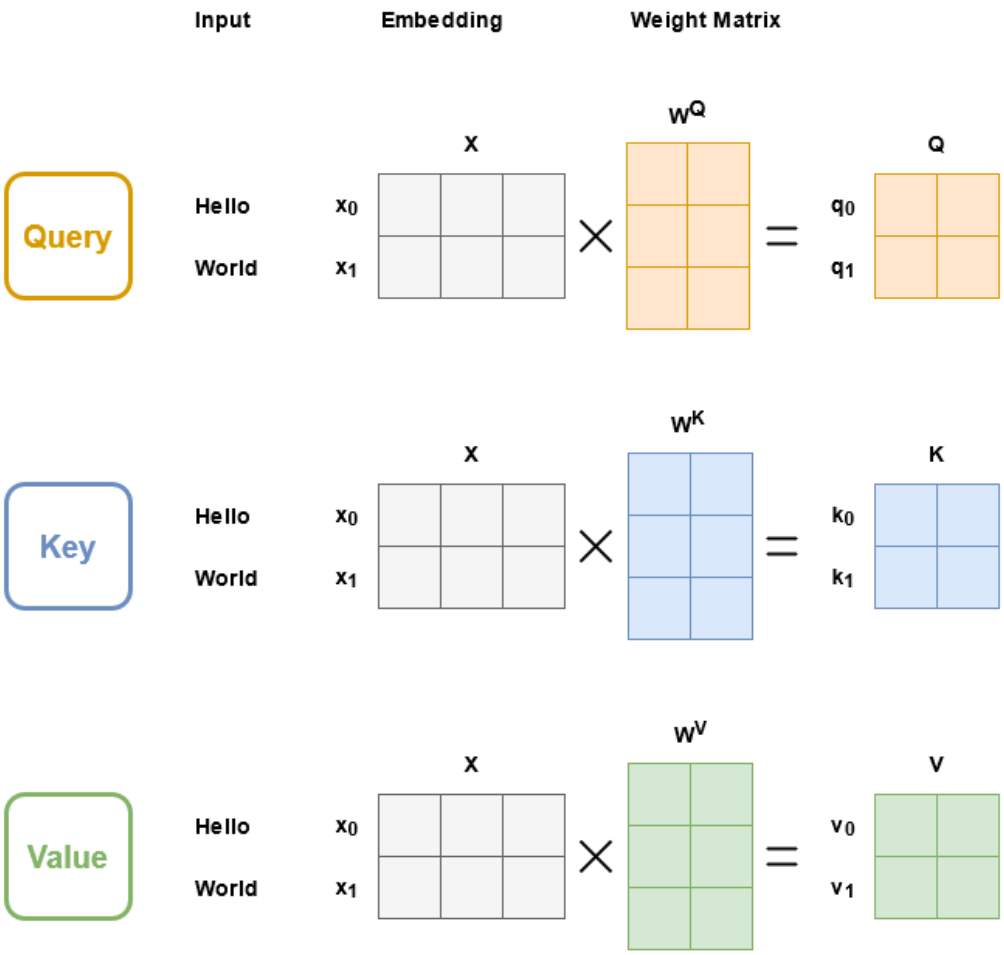

Figure 4.5: Calculate query, key, and values in self-attention

Figure 4.6 shows the calculations of the matrices scores and output. Matrix Q is multiplied with $K^\top$ and divided by $\sqrt{d_k}$. The dimension of the key vector is denoted as d_k. The Softmax function given in equation 4.6 is applied on every row of the previously calculated matrix, which results in the scores matrix. The output is calculated by multiplying the matrix scores with the matrix V. Each

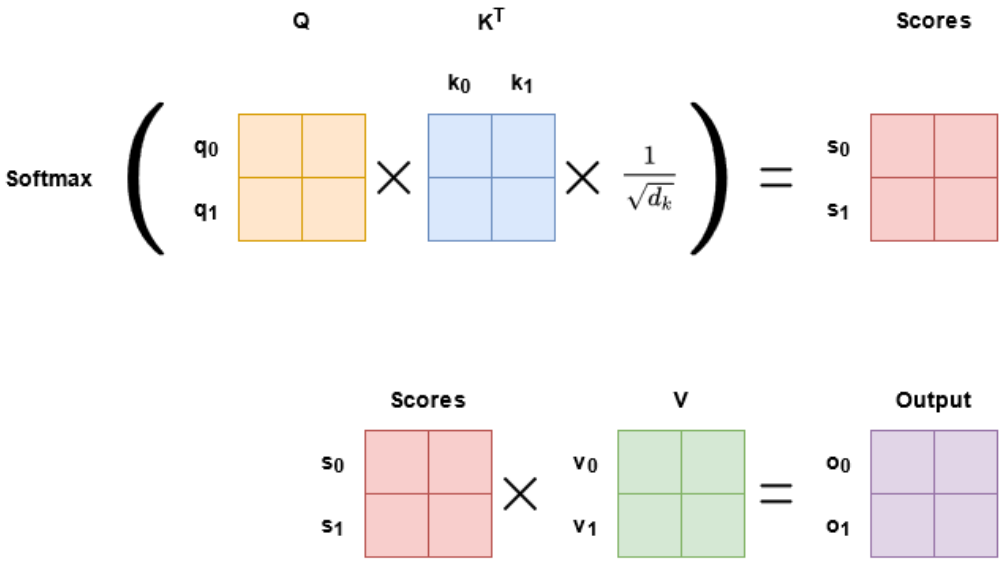

Figure 4.6: Calculate scores and output

row of the output matrix consists of the attention weights of the corresponding input. For example, the attention weights of input 0 are in row 0.

$$\text{Softmax}(\mathbf{x}) = \frac{e^{\mathbf{x}}}{\sum_i e^{\mathbf{x}_i}} \tag{4.6}$$

The complete calculation of attention can be written in a single equation which is given in 4.7.

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{4.7}$$

4.2.2 Transformer Architecture

In 2017, Vaswani et al. (2017) proposed a new architecture called transformer in the paper "Attention Is All You Need". Transformers are based on attention mechanisms, which allow for more parallelisation, resulting in faster training time. The architecture of transformers is illustrated in figure 4.7.

Input

The input of the transformer is first transformed into input embeddings. In NLP tasks, the input is a word sequence, for example, a sentence or short text. Each word of the input is transformed into a word embedding. A positional encoding is

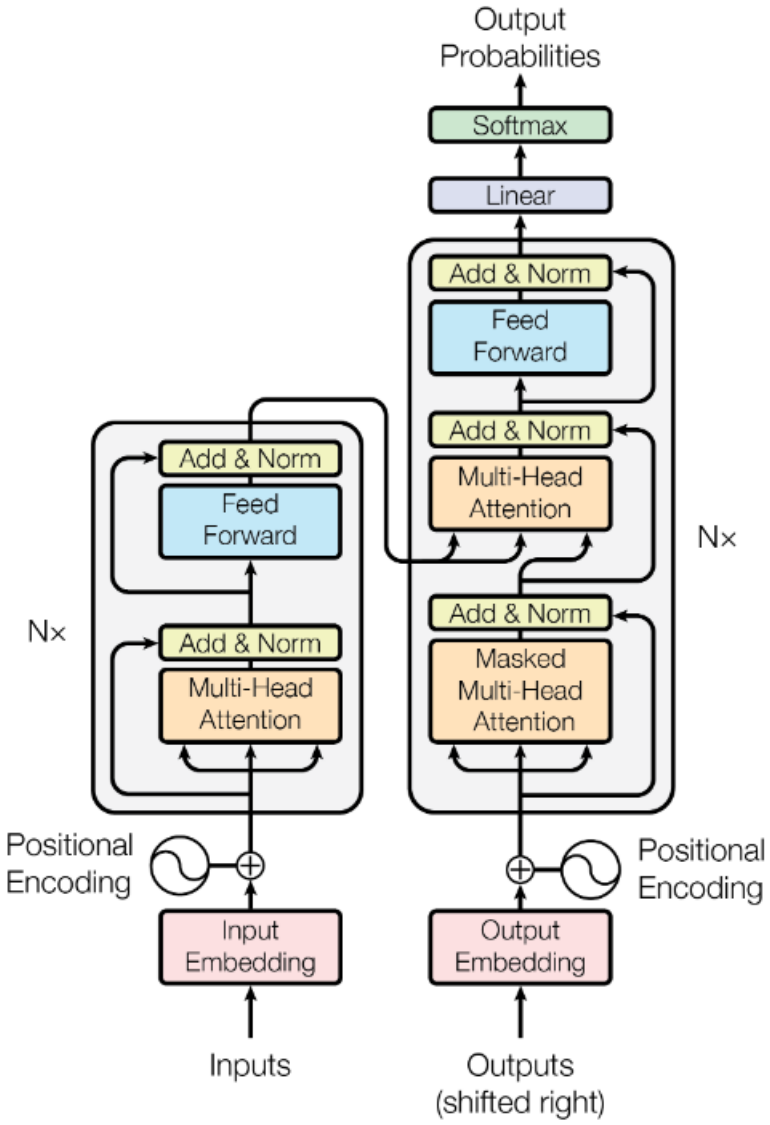

Figure 4.7: Transformer architecture (source: Vaswani et al. 2017)

added to the input embedding so that the transformer architecture can remember the input sequence order. The positional encoding is a vector with the same dimension as the input embedding.

Encoder Stack

The encoder stack consists of N layers. The structure of all layers is identical. An encoder layer consists of the two sublayers multi-head attention and fully connected

feed-forward network, which is labelled “Feed Forward” in figure 4.7. Each sublayer additionally has a normalisation layer, which is labelled “Add & Norm” in figure 4.7.

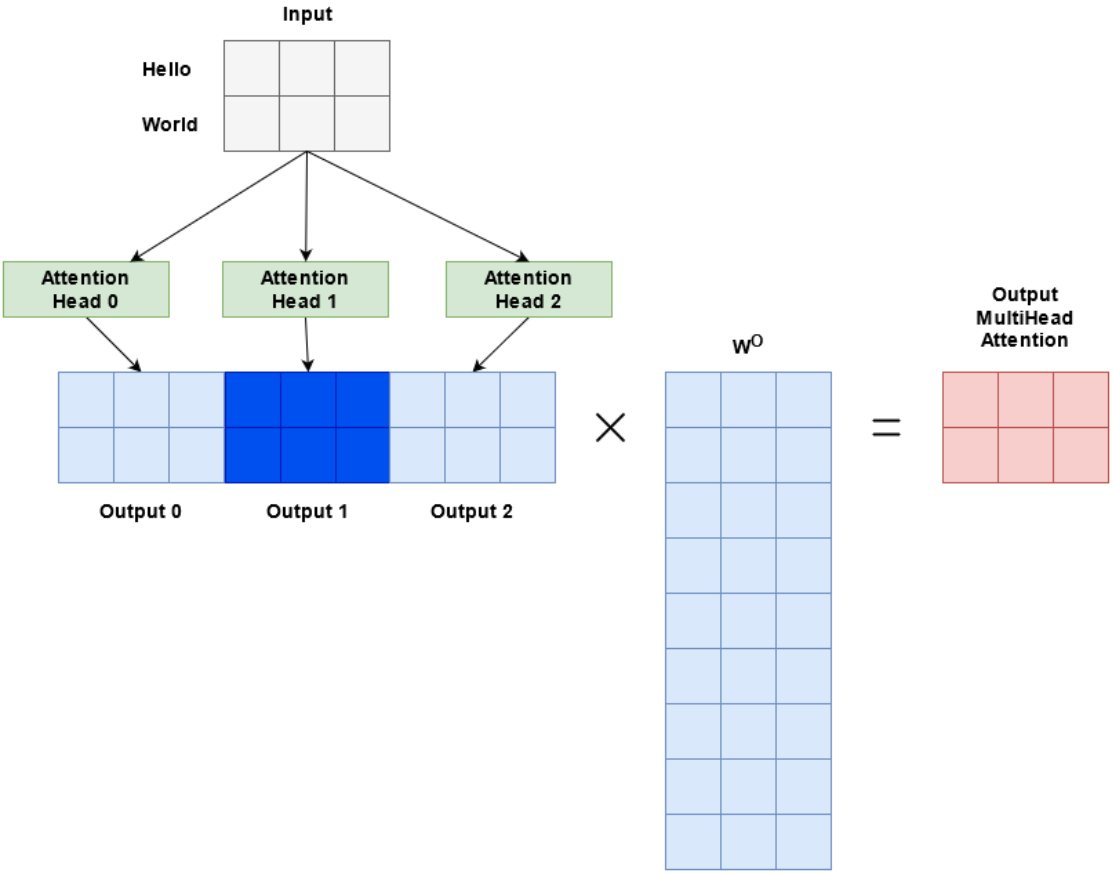

Figure 4.8: Calculation of multi-head Attention

Multi-head attention means that instead of a single self-attention head, h identical self-attention heads are used. However, the weights in each head are different. Figure 4.8 visualises the calculation of multi-head attention for the input sequence “Hello World” with three heads. The input sequence is processed in parallel by each head. The outputs of all heads are concatenated, which is then multiplied by the weight matrix W^O. According to Vaswani et al. (2017), the calculation for

MultiHead is given by equation 4.8.

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{4.8}$$

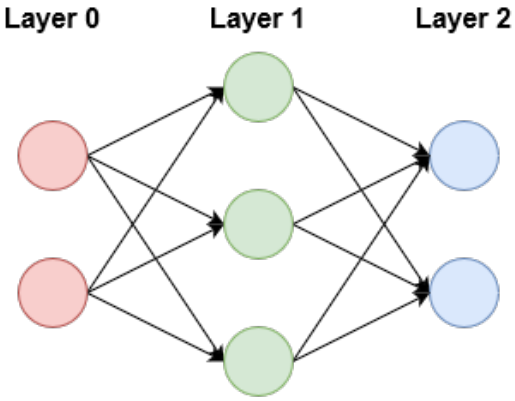

Figure 4.9: Fully connected feed-forward neural network

The second sublayer of the encoder is a fully connected feed-forward neural network. For better comprehension, a brief explanation about neural networks is given here. However, neural networks are complex, and it is not possible to go into detail here. Generally, a neural network consists of multiple neurons arranged in layers. Each neuron can have multiple connections to other neurons. In a fully connected feed-forward neural network, each neuron is connected to every neuron in the subsequent layer. An example for a fully connected feed-forward neural network is shown in figure 4.9. A neuron's input is the sum of all incoming connection weights multiplied by the output of the connected neuron. An activation function is applied to the input of the neuron. The result is the output of the neuron. A popular activation function is rectified linear unit (ReLU) (Fukushima 1980), which is given in equation 4.9.

$$\varphi(x) = \max(0, x). \tag{4.9}$$

In the transformer architecture, the same fully connected feed-forward neural network is applied individually to each input embedding of the input sequence (Vaswani et al. 2017). The neural network is given by equation 4.10. The neural

network consists of two linear transformations. After the first linear transformation, the ReLU activation function is applied.

$$\mathrm{FFN}(\mathbf{x}) = \max(0, \mathbf{x}W_1 + \mathbf{b}_1)W_2 + \mathbf{b}_2 \tag{4.10}$$

In the normalisation layer, the sublayer's input is added to its output and then normalised. The result is the input of the next sublayer.

Decoder Stack

The decoder stack is very similar to the encoder stack and also has N identical layers. The decoder layer consists of the three sub-layers masked multi-head attention, encoder-decoder attention (referred to as multi-head attention in figure 4.7), and a fully connected feed-forward network. The feed-forward network has the same structure as in the encoder.

In the masked multi-head attention layer, all subsequent inputs are masked for the input at position i. For example, if the sequence "Hello World" is processed and the attention head calculates the attention weights for input "Hello", then the subsequent input "World" is masked and not used in the calculation. Thus, only those inputs that the decoder has already processed at this point are used for the calculation of the attention weights. The masking is done by setting the score to $-\infty$ for the masked inputs before the softmax function is applied.

In the encoder-decoder attention, only the query is calculated with the input of the sublayer. Key and value are used from the last encoder. Hence, the name encoder-decoder attention.

Output

A linear transformation and the softmax function are applied after the decoder stack to calculate the predicted next-token probabilities from the decoder output.

4.2.3 BERT

BERT is a language representation model and was developed by Devlin et al. (2019). BERT is the first language representation model with deep bidirectional representations in the layers, which allowed it to achieve new state-of-the-art results in 11 tasks. The model architecture of BERT is a multi-layer bidirectional transformer encoder with L layers, the hidden size H and A self-attention heads (Devlin et al. 2019). The two original models, BERTbase and BERTlarge, use $L = 12$, $H = 768$, $A = 12$, and $L = 24$, $H = 1024$, $A = 16$, respectively. BERT

uses only the encoder part of the transformer architecture. The encoder part uses bidirectional self-attention, which allows BERT to understand the context on both sides of a token.

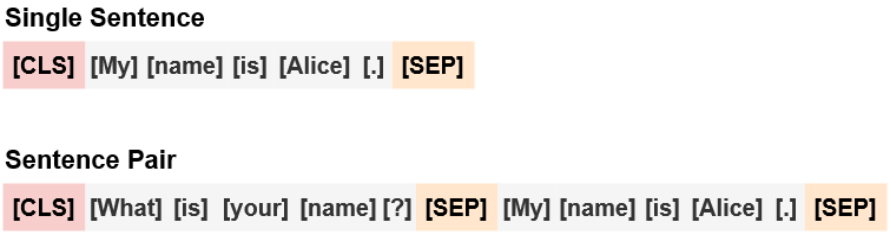

Figure 4.10: Example token sequences for BERT

The input of BERT is a token sequence with a maximum length of 512. The token sequence consists of either a single sentence or a sentence pair. For example, a sentence pair is a question and its corresponding answer. Sentence here means a coherent piece of text and not a linguistic sentence. Each token sequence begins with the special token [CLS], and each sentence ends with the separator token [SEP]. Two examples for token sequences are visualised in figure 4.10.

BERT was pre-trained with two unsupervised tasks: masked language modelling (MLM) and next sentence prediction (NSP). MLM, also referred to as Cloze task (Taylor 1953) in the literature, is used to train the bidirectional representation. In this task, some tokens are masked, and the machine learning algorithm must predict the token. For example, in the sentence "My name is Alice." the token "name" is masked, so the input of the machine learning algorithm is "My [MASKED] is Alice", and the correct output is "name". For the training of BERT 15% of the input tokens are randomly selected. With a probability of 80%, a selected token is replaced by the token [MASK]. 10% of the time, it is replaced by a random token and 10% of the time, the token is not changed. NSP is used to learn the relationship between two sentences. In training, 50% of the time sentence A is followed by the correct sentence B with label *IsNext* and 50% of the time sentence A is followed by a random sentence from the corpus with label *NotNext*. There are already numerous pre-trained BERT models that have been trained with texts from different languages. The training of a BERT model takes several days using tensor processing units (TPUs), depending on the size of the corpus.

The pre-trained architecture can be used for specific tasks like question answering or sentiment analysis by adding at least one additional output layer and fine-tuning all parameters with labelled data. For example, a dropout layer and a

linear layer with an output dimension of one can be added to solve a regression task. Depending on the data size, the training of the specific task usually takes a maximum of one hour on a TPU and is very fast compared to pre-training. Because so few modifications are required, pre-trained BERT models are suitable for a wide variety of natural language processing tasks.

4.2.4 Experiments

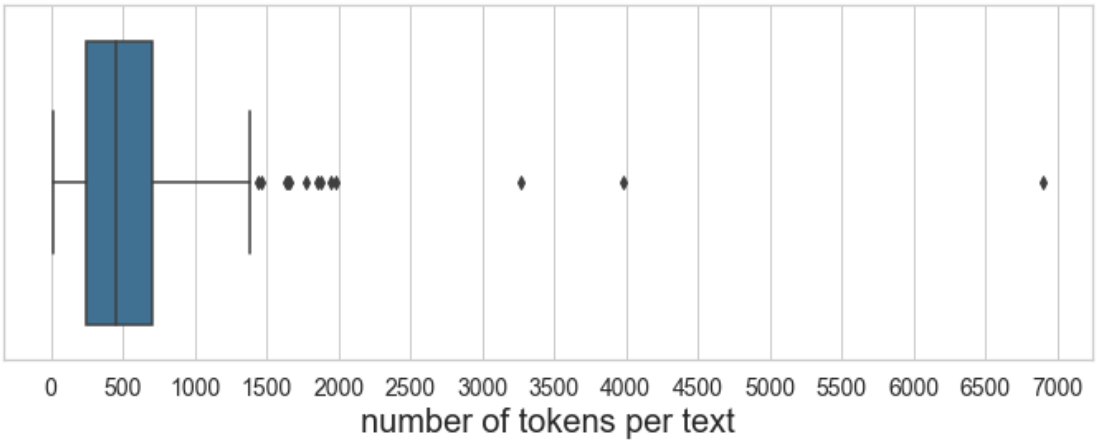

Figure 4.11: Number of tokens per article text

The BERT model allows a maximum of 512 tokens per input, including the two special tokens to mark the start and end of the input sequence, which restricts the text length to 510 tokens per article. Figure 4.11 shows the distribution of the text length of articles. The text length varies from 17-6,902 tokens, with an average text length of 525 tokens. Only three articles have more than 3,000 tokens.

The texts are grouped into three categories: short texts with 100 tokens or less, medium texts with a length up to 510 tokens, and long texts that exceed the limit of 510 tokens. Figure 4.12 shows the percentage of articles for each category. 42.4% of all articles exceed the limit of 510 tokens, whereas 14.8% of all articles are very short. Most articles have an additional description summarising the text. This description contains up to 83 tokens. If the text is prepended with the description, only 11.4% of all articles consist of 100 or fewer tokens.

The article texts must be preprocessed to fit the maximum length of 510 tokens. Sun et al. (2020) proposes three methods to truncate long texts: head-only uses the first 510 tokens, tail-only the last 510 tokens, and head + tail uses the first

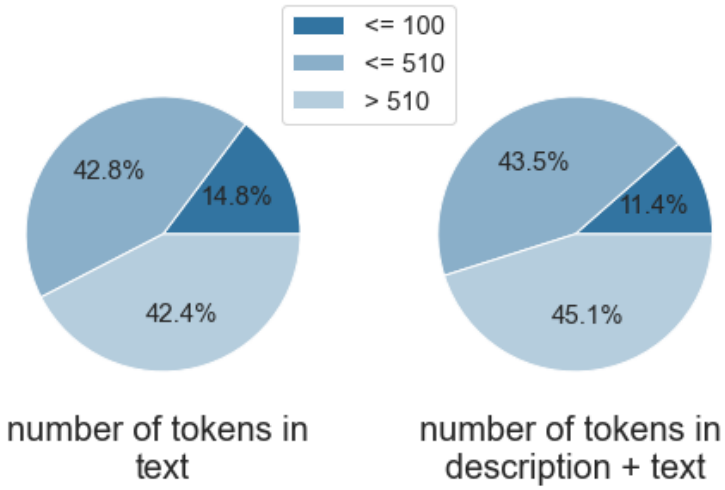

Figure 4.12: Per cent of articles with ≤ 100, ≤ 510, > 510 tokens per text (left) and per text + description (right)

128 and the last 382 tokens. The month of pregnancy was used as the label for training. Since this is a regression problem, the labels were linearly scaled from 2-11 to 0-1 with min-max normalisation.

A suitable learning rate and number of epochs were determined by hyperparameter optimisation with W&B[5]. The learning rate is $1e^{-4}$, and the model was trained for 40 epochs. Since only a few data are available, the training was evaluated with stratified 5-fold cross-validation. The performance of BERT is measured with RMSE.

The Python library SimpleTransformers[6] was used for creating and training the BERT model. SimpleTransformers adds a dropout layer and linear layer to the BERT architecture to solve the regression task. SimpleTransformers uses the PyTorch[7] implementation for both layers. The PyTorch documentation states that a dropout layer replaces input elements with 0 with the probability p and scales the output by factor $1/(1-p)$ during training. SimpleTransformers uses $p = 0.1$. The linear transformation $\mathbf{y} = W^\top \mathbf{x} + \mathbf{b}$ is applied in the linear layer. SimpleTransformers uses inputs with dimension 768, the output has dimension one.

There are already many pre-trained BERT models. Two models can be used for German texts. Devlin et al. (2019) trained the model bert-base-multilingual-

[5]wandb.ai
[6]simpletransformers.ai
[7]https://pytorch.org/

cased with Wikipedia articles from 104 languages. The German company deepset[8] trained a purely German BERT model, bert-base-german-cased. The German Wikipedia, the OpenLegalData data set[9] and news articles were used for the training (deepset 2021). The OpenLegalData data set contains 100,000 German court decisions and 444,000 citations. Since deepset could achieve better evaluation results with bert-base-german-cased than bert-base-multilingual-cased, the German model is used in this work.

Sun et al. (2020) achieved better results by further pre-training the BERT model with in-domain data on the MLM task. Since the articles on pregnancy use particular terms and the texts differ less in the terms than is the case, for example, with an article on sports and one on politics, this work investigates whether further pre-training also leads to better results in determining the month of gestation. The bert-base-german-cased model was further pre-trained with *swissmom's* article texts on the MLM task. All articles were divided into sentences using the library spaCy. The model was further trained for five epochs with a learning rate of $5e^{-5}$. The library HuggingFace Transformers[10] was used to implement the training.

Results

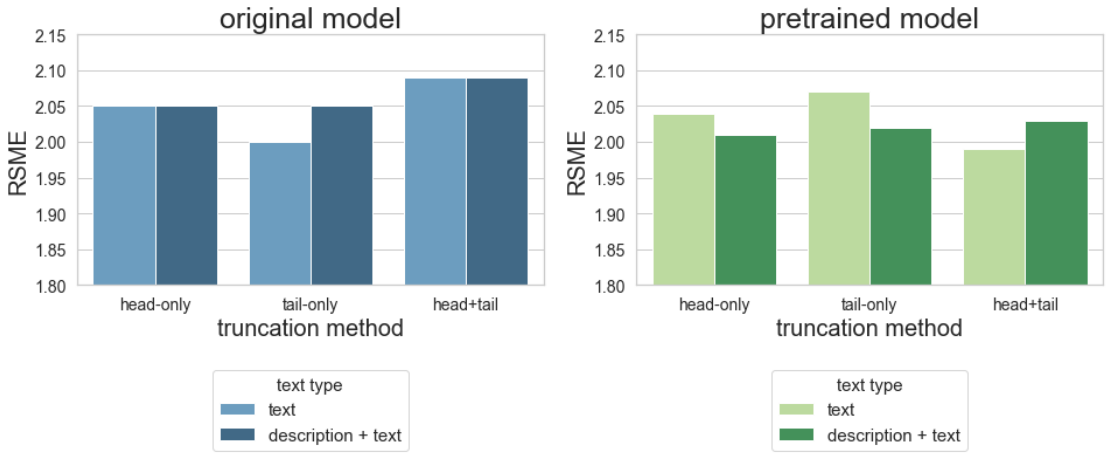

Figure 4.13: RMSE for BERT experiments

[8]https://deepset.ai
[9]http://openlegaldata.io/research/2019/02/19/court-decision-dataset.html
[10]https://huggingface.co/transformers

The three truncation methods were applied to the article text and the article text prepended with the article's description (description + text). All six preprocessed data sets were trained with the original bert-base-german-cased model and the further pre-trained model. The training results are shown in table 4.2. Additionally, the results are visualised as barplots in figure 4.13. A separate plot was created for the results of the original model and the further pre-trained model. The bars are grouped by the truncation method within each plot. Within a group, the results for text and description + text are shown. The y-axis shows the RMSE and is cut off below 1.8 for better visualisation of the results.

The absolute differences between the results are very small with a maximum difference of 0.1, which means that no clear statement can be made as to which experiment performed best. The pre-trained model performed slightly better than the original BERT model except for the truncation method tail-only combined with the data set containing only the article text. It would have been expected that all experiments improved at least slightly when trained on the pre-trained model. There is no indicator why the truncation-method tail-only without the description data performed worse than the same experiment trained on the original model. The best result with an RMSE of 1.99 is achieved with the further pre-trained model, the truncation method head + tail and without using the description of articles.

truncation method	original BERT	further pre-trained
text		
head-only	2.05	2.04
tail-only	**2.00**	2.07
head + tail	2.09	**1.99**
description+text		
head-only	2.05	2.01
tail-only	2.05	2.02
head + tail	2.09	2.03

Table 4.2: RMSE for BERT experiments

4.3 Discussion

The difficulties of both experiments are that very little data is available, and the texts to be classified are very similar, as all texts are about pregnancy and birth. This similarity makes it more difficult to predict the month of gestation based on

the text content than it is, for example, to predict categories such as sports and politics in news articles. Both experiments used different approaches. The TF-IDF experiment tries to determine the month of pregnancy by keyword extraction. This approach is more straightforward than the context-based approach of the BERT experiment. The BERT model has already been pre-trained with general German texts to understand the context of words within a short text.

The TF-IDF experiment compared the two complex preprocessing methods stemming and lemmatisation, with the more straightforward approach of removing stop-words to extract better keywords. Better results could be achieved with the simple stop-word removal approach. The best result with a RMSE of 2.16 months is achieved with stop-word removal, 10,000 features and ridge regression.

The BERT experiment compared different truncation methods to deal with BERT's problem of handling text longer than 510 tokens. Further pre-training of the original BERT model with in-domain data from *swissmom* was also conducted. The best result with a RMSE of 1.99 months is achieved with the truncation method head + tail and further in-domain pre-training. BERT outperformed TF-IDF by 0.17 months.

According to section 3 the span of relevant weeks of pregnancy is 11 weeks or 2.75 months. The results from TF-IDF model and BERT model are both below this span.

5 Summary and Outlook

This section will give a short summary of the work done. Furthermore, suggestions are made to improve the performance of the BERT model for predicting the month of pregnancy. Possibilities to improve the in-domain pertraining of BERT are presented. An extension of the recommender system is also briefly mentioned.

5.1 Summary

Section 1 discussed the particular challenges that occur with *swissmom's* data when designing a recommender system. Since 90% of users are visiting the website for the first time, a recommender system must deal with the cold start problem. Another complication that was mentioned is that the interests of pregnant users are dependent on the user's phase of pregnancy, and the interests, therefore, change continuously. At the beginning of pregnancy, a user is interested in, among other things, complications that can occur during the first months. The birth process is not relevant for most users at this time. If the user is at the end of pregnancy, the interests are the opposite.

Comprehensive literature research on recommender systems was conducted and described in section 2. The four basic models and the differences between implicit and explicit feedback were explained. Context-aware recommender systems were also considered in the course of the research and described in this section.

An exploratory data analysis was performed on *swissmom's* data and the findings were described in section 3. The Google Analytics data from 7 February until 7 March 2021 containing the user-item interactions was enriched with the week of gestation and category of articles. The data showed that 75% of all users interacted with three or fewer articles in the observed period. This confirmed that we need to deal with the cold start problem when designing the concept of the recommender system.

There are a total of six categories to which an article can be assigned. The analysis showed that the majority of users read articles from a maximum of two different categories. From all categories, "pregnancy" is the most popular one. The investigation of the pregnancy week of articles suggested that users are most interested

in articles where the assigned week of gestation is on average no more than 11 weeks apart. This means that users prefer to look at articles that are relevant in the same pregnancy phase.

The user's week of gestation is unknown in the data. Therefore, it was assumed based on the first article viewed by the user that has a week of gestation assigned. The findings indicated that there is a moderate positive correlation between the hypothetical week of pregnancy of a user and the actual week of pregnancy of an article. Furthermore, this analysis also suggested that users are interested in articles that belong to the same phase of pregnancy. These results also support the initial statement that there are pregnancy-related temporal changes in interest which must be addressed in order to generate sound recommendations.

The techniques for recommender systems presented in the state-of-the-art research were evaluated for their suitability to solve the challenges based on the findings of the data analysis. Context-aware recommender systems are not applicable because they need the unknown week of pregnancy of users as context information to predict sound recommendations. Collaborative and content-based filtering models are not well-suited to solve the challenge with the cold start problem because they rely on historical user-item interactions. However, this data is not available for most users that visit *swissmom's* website. Moreover, these two models cannot handle the temporal changes in interest when the user's gestational stage is unknown.

On the other hand, case-based recommender systems are very well suited to deal with the cold-start problem. Therefore, the proposed concept for a recommender system was a case-based model that recommends articles from the same category as the article that is currently being viewed. If this article is about pregnancy, the assigned week of pregnancy is also used for generating recommendations. In this case, articles that belong to the same pregnancy phase as the currently visited article are recommended. In doing so, the challenge with the temporal changes of interest is also solved.

The proposed recommender system needs to know in which week of gestation each article about pregnancy is relevant. However, the analysis showed that this information is only known for 31% of all articles on pregnancy. Experiments were conducted to predict an article's month of pregnancy, which was described in section 4. Due to the limited data available, the month was determined rather than the week. The regression task was evaluated with 5-fold cross-validation. RMSE was used to determine the performance.

In the first part of the experiments, the keyword-based TF-IDF model was used. The preprocessing methods stop-word removal, stemming, and lemmatisation were

applied and compared. It was shown that the best results were not achieved with the complex methods stemming and lemmatisation but with stop-word removal. The experiment also showed that 10,000 features are enough to achieve good results. A RMSE of 2.16 months was achieved with ridge regression, which is below the interest span of 2.75 months determined in the data analysis.

In the second part of the experiments, the context-based BERT model was used. Since BERT allows texts with a maximum length of 510 tokens, the three truncation methods head-only, tail-only and head + tail were compared. In addition, BERT was further pre-trained with in-domain data on the unsupervised MLM task. Slightly better results could be achieved with further in-domain pre-training. The best result was achieved with the truncation method head + tail and resulted in a RMSE of 1.99 months. The BERT model performed better than the TF-IDF model. In the context of *swissmom*, it was shown that with little training data and very subtle differences between the different classes, the context of the words matters.

5.2 Outlook

The experiments using TF-IDF and BERT showed that the context-based approach was better suited for predicting an article's month of pregnancy. The limited data available for predicting the month was a significant drawback. Increasing the size of the training data set could help improve BERT's performance. One possibility would be to use more data sets with German articles on pregnancy and birth if such data sets exist and are available. For predicting the articles' month of pregnancy, these articles must be assigned to a week or month of pregnancy. A second possibility to increase the size of the labelled training data set would be different data augmentation techniques. Sennrich, Haddow, and Birch (2016) suggest back-translation. In this process, data sets from other languages are translated into the target language. If there are available data sets with labelled articles on pregnancy in, for example, English, these could be translated into German and used as additional training data. Easy data augmentation (EDA), proposed by Wei and Zou (2019), modifies the existing training texts through the five operations synonym replacement, random insertion, random swap, and random deletion. Wei and Zou have evaluated EDA in the context of convolutional neural networks (CNN) and recurrent neural networks (RNN). However, this method could also be well suited to generate more training data for BERT. Kumar, Choudhary, and E. Cho (2021) investigated the use of pre-trained seq2seq models, such as BART, to generate synthetic training data and achieved even better results than with EDA and back-translation. BART was proposed by Lewis et al. (2019) and is a

pre-trained model combining bidirectional and auto-regressive transformers.

Further pre-training of BERT with unlabelled data led to slightly better results in learning the articles' month of pregnancy. Again, performance could potentially be improved by using a more extensive data set. Since unlabelled data is needed here and this type of data is more available as compared to labelled data, this option is more accessible to implement than increasing the size of the labelled training data set. Utilising back-translation, the use of non-German texts on pregnancy is also possible.

Predicting the month of pregnancy for all pregnancy-related articles creates the basis for designing a case-based recommender system. When a user reads an article, the articles about pregnancy can be filtered to only include articles from the relevant month of gestation. This selection must be further narrowed down by, for example, ranking all relevant articles with a ranking algorithm and choosing the top k articles. Often, the ranking of texts in recommender systems is done by similarity metrics. If recommendations are to be generated for article X, the k articles with the greatest similarity to X are recommended. When representing texts with the TF-IDF model, the vector representation of the texts can then be used to calculate the similarity, for example, the cosine similarity.

A prototype of the proposed recommender system can be evaluated online with A/B testing. In this testing setup, a user visiting the website is randomly assigned to group A or group B. Typically, random assignment is done with a probability of 50%. Users in group A will receive recommendations from the recommender system under evaluation, and users in group B will receive recommendations from the currently used recommender system. In this way, it can be tested whether the original or the new recommender system is better received by the users.

Providing users with the possibility to enter their due date on *swissmom's* website would add crucial information to the data. With this information available, experiments using collaborative filtering approaches could be conducted. If the experiments are successful, a hybrid recommender system could be implemented combining the proposed case-based model with the collaborative filtering method.

Bibliography

ACM RecSys (2021). *RecSys – ACM Recommender Systems*. URL: https://recsys.acm.org/recsys21/ (visited on 04/15/2021).

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural Machine Translation by Jointly Learning to Align and Translate". In: ICLR. URL: http://arxiv.org/abs/1409.0473 (visited on 07/03/2021).

Burke, Robin (Nov. 1, 2002). "Hybrid Recommender Systems: Survey and Experiments". In: *User Modeling and User-Adapted Interaction* 12.4, pp. 331–370. URL: https://doi.org/10.1023/A:1021240730564 (visited on 07/13/2021).

Campos, Pedro, Fernando Rubio, and Iván Cantador (Feb. 1, 2014). "Time-aware recommender systems: A comprehensive survey and analysis of existing evaluation protocols". In: *User Modeling and User-Adapted Interaction* 24, pp. 67–119. URL: https://www.researchgate.net/publication/257671605_Time-aware_recommender_systems_A_comprehensive_survey_and_analysis_of_existing_evaluation_protocols/stats.

CH Regionalmedien AG (2021a). *Facebook swissmom.ch*. URL: https://www.facebook.com/pages/category/News---Media-Website/swissmom.ch/about/ (visited on 04/16/2021).

— (2021b). *swissmom.ch*. swissmom.ch. URL: https://www.swissmom.ch/impressum/ (visited on 04/16/2021).

deepset (2021). *deepset - Open Sourcing German BERT*. URL: https://deepset.ai/german-bert (visited on 06/19/2021).

Deshpande, Mukund and George Karypis (Jan. 1, 2004). "Item-based top-N recommendation algorithms". In: *ACM Transactions on Information Systems* 22.1, pp. 143–177. URL: https://doi.org/10.1145/963770.963776 (visited on 07/03/2021).

Devlin, Jacob et al. (May 24, 2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. URL: http://arxiv.org/abs/1810.04805 (visited on 05/24/2021).

Drucker, Harris et al. (1996). "Support Vector Regression Machines". In: *Advances in Neural Information Processing Systems* 9, pp. 155–161. URL: https://proceedings.neurips.cc/paper/1996/file/d38901788c533e8286cb6400b40b386d-Paper.pdf.

Fawcett, Tom (Jan. 1, 2004). "ROC Graphs: Notes and Practical Considerations for Researchers". In: *Machine Learning* 31, pp. 1–38. URL: https://www.

researchgate.net/publication/284043217_ROC_Graphs_Notes_and_Practical_Considerations_for_Researchers/citations.

Felfernig, Alexander, Gerhard Friedrich, et al. (2011). "Developing Constraint-based Recommenders". In: *Recommender Systems Handbook*. Springer, pp. 187–216.

Felfernig, Alexander, Michael Jeran, et al. (May 24, 2013). "Toward the Next Generation of Recommender Systems: Applications and Research Challenges". In: *Smart Innovation, Systems and Technologies*. Vol. 24, pp. 81–98. ISBN: 978-3-319-00374-0. URL: https://www.researchgate.net/publication/233886182_Toward_the_Next_Generation_of_Recommender_Systems_Applications_and_Research_Challenges.

Fukushima, Kunihiko (Apr. 1, 1980). "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position". In: *Biological Cybernetics* 36.4, pp. 193–202. URL: https://link.springer.com/article/10.1007/BF00344251 (visited on 07/28/2021).

Herlocker, Jonathan L. et al. (Jan. 1, 2004). "Evaluating collaborative filtering recommender systems". In: *ACM Transactions on Information Systems* 22.1, pp. 5–53. URL: https://doi.org/10.1145/963770.963772 (visited on 07/14/2021).

Hoerl, Arthur and Robert Kennard (1970). "Ridge Regression: Biased Estimation for Nonorthogonal Problems". In: *Technometrics* 12, pp. 55–67.

Hofmann, Thomas (2004). "Latent Semantic Models for Collaborative Filtering". In: *ACM Transactions on Information Systems* 22.1, pp. 89–115. URL: https://dl.acm.org/doi/10.1145/963770.963774.

Joachims, Thorsten (Jan. 1, 1998). "Text Categorization with Support Vector Machines". In: *Proc. European Conf. Machine Learning (ECML'98)*. URL: https://www.researchgate.net/publication/28351286_Text_Categorization_with_Support_Vector_Machines.

Karim, Raimi (Nov. 18, 2019). *Illustrated: Self-Attention*. Illustrated: Self-Attention. URL: https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a (visited on 06/12/2021).

Klahold, André (Feb. 26, 2009). "Empfehlungssysteme: Recommender Systems - Grundlagen, Konzepte und Lösungen". In: 1. Edition. Wiesbaden: Vieweg + Teubner Verlag, pp. 1–2.

Kumar, Varun, Ashutosh Choudhary, and Eunah Cho (Jan. 31, 2021). *Data Augmentation using Pre-trained Transformer Models*. URL: http://arxiv.org/abs/2003.02245 (visited on 06/22/2021).

Lewis, Mike et al. (Oct. 29, 2019). *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. URL: http://arxiv.org/abs/1910.13461 (visited on 06/22/2021).

Lops, Pasquale, Marco de Gemmis, and Giovanni Semeraro (Jan. 1, 2011). "Content-based Recommender Systems: State of the Art and Trends". In: *Recommender Systems Handbook*, pp. 73–105. URL: https://www.researchgate.net/publication/226098747_Content-based_Recommender_Systems_State_of_the_Art_and_Trends.

Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning (Sept. 20, 2015). *Effective Approaches to Attention-based Neural Machine Translation*. URL: http://arxiv.org/abs/1508.04025 (visited on 07/03/2021).

McGinty, Lorraine and James Reilly (2011). "On the Evolution of Critiquing Recommenders". In: *Recommender Systems Handbook*. Springer, pp. 419–453.

Mikolov, Tomas, Quoc V. Le, and Ilya Sutskever (Sept. 17, 2013). *Exploiting Similarities among Languages for Machine Translation*. URL: https://arxiv.org/abs/1309.4168v1 (visited on 07/22/2021).

Netflix, Inc. (2021). *Netflix Prize*. URL: https://www.netflixprize.com/ (visited on 04/15/2021).

Pazzani, Michael and Daniel Billsus (June 1, 1997). "Learning and Revising User Profiles: The Identification of Interesting Web Sites". In: *Machine Learning* 27.3, pp. 313–331. URL: https://doi.org/10.1023/A:1007369909943 (visited on 04/29/2021).

Porter, M. (1980). "An algorithm for suffix stripping". In: *Program* 40, pp. 211–218.

Saif, Hassan, Miriam Fernandez, and Harith Alani (May 26, 2014). "On Stopwords, Filtering and Data Sparsity for Sentiment Analysis of Twitter". In: The 9th International Conference on Language Resources and Evaluation. Iceland. URL: http://www.lrec-conf.org/proceedings/lrec2014/pdf/292_Paper.pdf.

Salton, G. and C. Buckley (1988). "Term-Weighting Approaches in Automatic Text Retrieval". In: *Information Processing and Management* 24, pp. 513–523. URL: https://ecommons.cornell.edu/bitstream/handle/1813/6721/87-881.pdf?sequence=1&isAllowed=y.

Sennrich, Rico, Barry Haddow, and Alexandra Birch (June 3, 2016). *Improving Neural Machine Translation Models with Monolingual Data*. URL: http://arxiv.org/abs/1511.06709 (visited on 06/22/2021).

Shani, Guy and Asela Gunawardana (Jan. 1, 2011). "Evaluating Recommendation Systems". In: *Recommender Systems Handbook*. Vol. 12, pp. 257–297.

Smyth, Barry and Paul McClave (2001). "Similarity vs. Diversity". In: *Case-Based Reasoning Research and Development*. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 347–361.

Su, Xiaoyuan and Taghi M. Khoshgoftaar (2006). "Collaborative Filtering for Multi-class Data Using Belief Nets Algorithms". In: *Proceedings of the International Conference on Tools with Artificial Intelligence (ICTAI'06)*. Pro-

ceedings of the International Conference on Tools with Artificial Intelligence (ICTAI'06). ISSN: 2375-0197, pp. 497–504.

Sun, Chi et al. (Feb. 5, 2020). *How to Fine-Tune BERT for Text Classification?* URL: http://arxiv.org/abs/1905.05583 (visited on 05/24/2021).

Taylor, Wilson L. (1953). "Cloze Procedure: A New Tool for Measuring Readability". In: *Journalism Bulletin* 30, pp. 415–433.

Ungar, L. and Dean P. Foster (1998). "Clustering Methods for Collaborative Filtering". In: *Proceedings of the Workshopon Recommendation Systems*. Proceedings of the Workshopon Recommendation Systems. URL: https://www.aaai.org/Papers/Workshops/1998/WS-98-08/WS98-08-029.pdf (visited on 07/27/2021).

Vaswani, Ashish et al. (Dec. 5, 2017). *Attention Is All You Need*. URL: http://arxiv.org/abs/1706.03762 (visited on 06/07/2021).

Wei, Jason and Kai Zou (Aug. 25, 2019). *EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks*. URL: http://arxiv.org/abs/1901.11196 (visited on 06/22/2021).

Yang, Zhe et al. (Jan. 1, 2016). "A Survey of Collaborative Filtering Based Recommender Systems for Mobile Internet Applications". In: *IEEE Access* 4. URL: https://www.researchgate.net/publication/303556519_A_Survey_of_Collaborative_Filtering_Based_Recommender_Systems_for_Mobile_Internet_Applications.

List of Figures

Acronyms

ARHR average reciprocal hit-rate. 18

BERT bidirectional encoder representations from transformers. 31, 39, 40, 46–52, 55, 56

CMS content management system. 9, 20

CNN convolutional neural networks. 55

EDA easy data augmentation. 55

FPR false-positive rate. 19

IDF inverse document frequency. 35

MLM masked language modelling. 47, 50, 55

NLP natural language processing. 36, 42

NLTK Natural Language Toolkit. 36

NSP next sentence prediction. 47

ReLU rectified linear unit. 45, 46

RMSE root mean squared error. 32, 38, 39, 49, 51, 52, 54, 55

RNN recurrent neural networks. 55

ROC receiver operating characteristic. 18, 19

SVR Support Vector Regression. 37–39

TF term frequency. 35

TF-IDF term frequency - inverse document frequency. 31–33, 35–37, 52, 54–56

TPR true-positive rate. 19

TPU tensor processing unit. 47, 48