
Ekstra Bladet Recommender Project

- Enhancing Ekstra Bladet's User Engagement with a
Machine Learning-Powered Recommender System for News
Applications -

Project Report
Group 1

Aalborg University
Electronics and IT



AALBORG UNIVERSITY

STUDENT REPORT

Electronics and IT

Aalborg University

<http://www.aau.dk>

Title:

Ekstra Bladet Recommender Project - Enhancing Ekstra Bladet's User Engagement with a Machine Learning-Powered Recommender System for News Applications

Theme:

Scientific Theme

Project Period:

Spring Semester 2024

Project Group:

1

Participant(s):

Anders Mazen Youssef

Bence Szabo

Louise Foldøy Steffens

Supervisor(s):

Andres Masegosa

Copies: 1

Page Numbers: 150

Date of Completion:

May 28, 2024

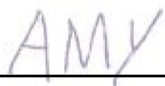
Abstract:

This report details the development of a machine-learning-powered recommender system to boost user engagement for Ekstra Bladet's mobile app. Developed using Agile methods, the modular system includes a mobile frontend, model training component, model-serving API, and a cloud database. The app tracks user behaviour, stores data, and employs LightFM for collaborative filtering to perform real-time article recommendations. Following configuration changes, the model achieved an AUC score of 0.9, effectively predicting user preferences. The system's design ensures scalability and easy integration of new models. Ethical concerns about personalised content and filter bubbles were addressed, and considerations for future work were documented. This project delivers a scalable, modular solution for personalised news recommendations.

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.

Preface

Aalborg University, May 28, 2024



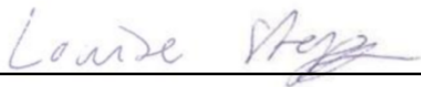
Anders Mazen Youssef

amyo21@student.aau.dk



Bence Szabo

bszabo21@student.aau.dk



Louise Foldøy Steffens

lfst21@student.aau.dk

Contents

Preface	iv
1 Introduction and Motivation	5
1.1 Initial problem description	7
2 State Of The Art	10
2.1 Mobile applications using recommender systems	10
2.1.1 Twitter	11
2.1.2 TikTok	11
2.2 Recommender systems in news	12
2.3 Recommender Systems	14
2.3.1 Content-based filtering	14
2.3.2 Collaborative filtering	15
2.3.3 Hybrid approach	16
2.4 Deployment of recommender systems - Model serving	17
2.5 Model evaluation	18
3 Analysis	21
3.1 Context - Ekstra Bladet's data and current recommender system for web	21
3.2 Course of action regarding MLOps	23
3.3 Course of action regarding recommender implementation	24

3.4 Problem statement	26
3.5 Requirements	26
4 Technology stack	28
4.1 Development environment	28
4.2 Application Technology	29
4.3 Recommender Technology	29
4.4 Potential off-the-shelf recommenders	30
4.4.1 DKN	31
4.4.2 LightFM	32
4.5 Database	33
4.6 System architecture	34
5 Implementation	35
5.1 System architecture in detail	35
5.2 The Ekstra Bladet News Recommendation Dataset (EBNeRD)	38
5.3 Agile workflow and collaboration with Ekstra Bladet	39
5.4 Sprint 1	42
5.4.1 Sprint Planning	42
5.4.2 Sprint Review	47
5.5 Sprint 2	48
5.5.1 Sprint Planning	49
5.5.2 Sprint Review	51
5.6 Sprint 3	53
5.6.1 Sprint Planning	53
5.6.2 Sprint Review	58
5.7 Sprint 4	59
5.7.1 Sprint Planning	59
5.7.2 Sprint Review	69
5.8 Sprint 5	71

5.8.1 Sprint Planning	71
5.8.2 Sprint Review	73
6 Quality Assurance	74
6.1 Evaluation of the system	75
6.1.1 Module cohesion	75
6.1.2 Module coupling	76
6.2 Code quality	77
6.3 Model Quality	79
6.4 User validation	82
7 End product	83
7.1 Mobile Application	83
7.1.1 Personalised News Feed	83
7.1.2 Article Interaction	84
7.1.3 Swipe-able Article Cards	84
7.1.4 User Behaviour Tracking	85
7.2 Model Serving	86
7.3 Model Training	86
7.4 Cloud Database	86
8 Discussion	88
8.1 Reflections on the development process	88
8.1.1 The product owner's reflections on the development process	91
8.2 Technical difficulties	91
8.3 Future work and scaling	92
8.3.1 The product owner's thoughts on future work	92
8.3.2 Leftover PBIs	93
8.4 Ethical concerns	96
9 Conclusion	99

Project Summary

The project, conducted in collaboration with the Danish tabloid newspaper Ekstra Bladet, aims to enhance user engagement on their mobile application by implementing a machine-learning-based recommender system. The motivation stems from Ekstra Bladet's need to retain users and increase engagement, given the trend towards consuming news on personalised platforms like TikTok and Twitter. Despite having a recommender system on their website, the mobile app lacks this feature, prompting the need for a more integrated and engaging mobile experience.

The report reviews current recommender systems used in mobile applications and the news industry, highlighting the effectiveness of content-based, collaborative, and hybrid filtering methods. It examines the deployment and evaluation of these systems, using examples from Twitter, TikTok, and The New York Times. The focus is on understanding the best practices and technologies to inform the design and implementation of Ekstra Bladet's recommender system.

The analysis explores Ekstra Bladet's data and current recommender system, noting that the existing model is underutilised on the web platform and absent on the mobile app. The project's problem statement emphasises the need for a highly modular system that supports a machine-learning-based recommender model to enhance user engagement through a personalised news feed on the mo-

bile application.

The chosen technology stack includes GitHub for version control and continuous integration, FastAPI for model serving, and proposes several types of off-the-shelf options for machine learning.

The implementation phase follows an Agile workflow, divided into multiple sprints. Each sprint focuses on different aspects of the system, from developing the mobile application and integrating the recommender model to refining the user interface and improving model performance. Key tasks include creating APIs for model predictions, training models on the Ekstra Bladet News Recommendation Dataset (EBNeRD), and ensuring the modularity and scalability of the system.

Quality assurance involves evaluating the system's modularity, code quality, and model performance. Techniques like static code analysis, unit testing, and user validation are employed to ensure the system meets the desired standards. The AUC score is used to assess model performance, with tools like TensorBoard providing visual representations of training progress.

The discussion reflects on the development process, technical challenges, and future work. It acknowledges the complexity of integrating a machine-learning model into a production environment and the importance of maintaining a scalable and flexible system. Future enhancements may include refining the recommendation algorithms, improving the user interface, and incorporating more advanced machine-learning models.

The report concludes with reflections on the project's progress and the necessary steps to continue developing and scaling the system. Future work will focus

on continuous improvement and ethical considerations to ensure the system remains effective and responsible in its user engagement strategies.

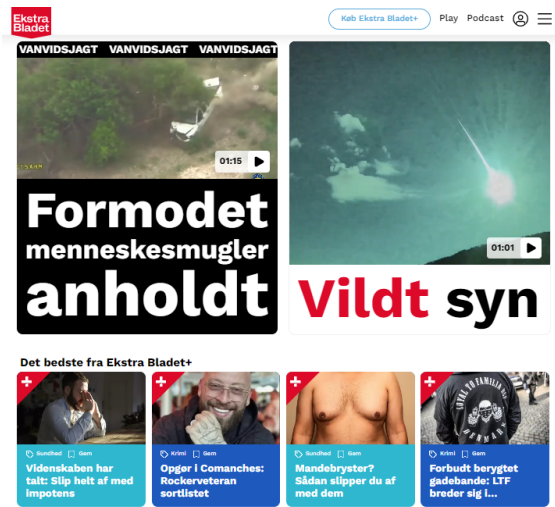
Chapter 1

Introduction and Motivation

This project is a collaboration with the Danish tabloid newspaper Ekstra Bladet, known for publishing content both in print and online. While Ekstra Bladet offers its articles through various mediums, including a mobile application, the app currently lacks several key features available on its website (e.g. platform-specific article layout and recommendations under articles) which is seen in figure 1.1. Users often find it inconvenient as it merely redirects to the browser when accessing articles, leading to its unpopularity. 9

Ekstra Bladet is one of the most visited news sites in Denmark. [38] Their primary source of revenue is ads and their premium subscription service, which means that if they cannot keep the users on the site, they do not earn as much per visitor. User engagement is vital for their revenue practices.

Considering that a significant percentage of young individuals now prefer consuming news content through highly personalised platforms like TikTok, Twitter, and Instagram [8], Ekstra Bladet's product developer, Kristoffer Hartwig, envisions the future of news consumption to involve delivering a more tailored user experience. 9 This entails leveraging machine learning models to predict the ar-



(a) The Ekstra Bladet website



(b) The Ekstra Bladet app

Figure 1.1: Ekstra Bladet's website and app

ticles a particular user is likely to find engaging, thus enhancing user retention and engagement within the app.

As there is a shift towards personalised news consumption, machine learning offers a powerful solution for delivering tailored user experiences. These systems have been proven to work, as on average, an intelligent recommender system delivers a 22.66% increase in user engagement. [13] By analysing patterns in user behaviour and preferences, machine learning models can predict which articles are likely to captivate individual readers. This predictive capability makes machine learning an ideal tool for news platforms aiming to enhance user engagement and retention. Through sophisticated algorithms, the system can adapt to evolving interests, ensuring users receive content that resonates with them on a personal level. This seamless integration of machine learning into news delivery is central to Ekstra Bladet's strategy for creating a more engaging app experience.

While a machine learning model in itself is a major component of the system,

with a dedicated team behind it such as Ekstra Bladet’s data science team, it is constantly undergoing improvements. This means that a focal point of this project should be to provide a system where parts are easily exchangeable. In broad terms, an overhauled news application should present relevant articles to app users predicted by a machine-learning model, using Ekstra Bladet’s article and user interaction data. To support this system, the app should be able to record and provide relevant data to developers, and this data should be used to improve future predictions made by a machine-learning model.

1.1 Initial problem description

During the initial meeting with Ekstra Bladet, which included their product developer, Kristoffer Hartwig and a member of their data science team, Trine Englund, it was established that they do not struggle with onboarding, but rather keeping their users engaged on the site. [9](#)

As of the writing of this report, Ekstra Bladet’s recommender system is only used in a small carousel section under a clicked article on the web platform. Although the data science team was not permitted to share their model, it was established that a proper recommender model could serve a much greater purpose than what it is currently used for. On the app, the recommender system is not used at all.

Based on the information gathered during the first meeting, Ekstra Bladet’s setup looks approximately like this:

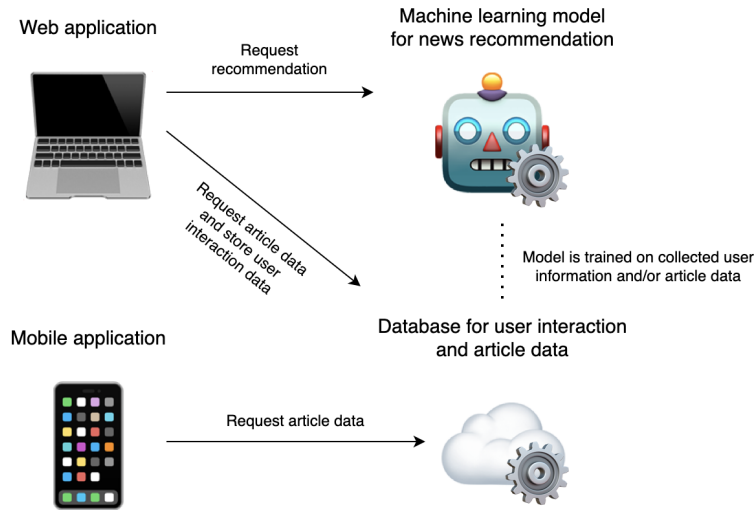


Figure 1.2: An approximate image of Ekstra Bladet's setup

The key takeaway from this figure is that the mobile application is not properly connected to the rest of the system. User interactions are not monitored, and the existing recommender model is not used. Given that the same machine-learning model should be used on both the web and in the mobile application, the importance of modularity is highlighted here. To summarise, the overarching goal of this project is to provide an improved overall system that is modular and scalable, and focuses on implementing a recommender technology to improve the experience on the mobile platform. As the figure [1.2](#) above suggests, the mobile application in this improved system should be on par with the web platform, using the same system components - thus taking advantage of a machine-learning-powered recommender to enhance user engagement.

The relevancy of the project's goal and Ekstra Bladet's interest in using machine-learning-powered recommenders is further emphasised by their launch of the RecSys Challenge 2024 shortly after the start of this project. [\[12\]](#) In this challenge, participants receive article and user interaction data in the form of a dataset

called the Ekstra Bladet News Recommendation Dataset (EBNeRD). Their task is to create the best-performing machine learning model.

Chapter 2

State Of The Art

Understanding and implementing state-of-the-art recommender systems is crucial for delivering a personalised user experience in line with current standards. This chapter explores the use of recommender systems in mobile applications and their specific application in the news industry. Additionally, it covers various types of recommender systems, their operational aspects (MLOps), and performance monitoring techniques. Establishing these fundamental concepts and exploring current applications should aid the development team in mapping out the most important requirements for a solution.

2.1 Mobile applications using recommender systems

From the initial meeting with the product owner at Ekstra Bladet, it was clear that their vision for their future on mobile platforms would be heavily inspired by popular social media platforms such as Twitter and TikTok. [9] This aligns well with a study [24] from 2023 that concluded that half of its participants, who were within the age range of 18 to 65, use social media as one of their main sources of news. Furthermore, 69% of the participants in this segment were within the age range of 18 to 29 years. The project's product owner also mentioned this demographic of young adults as a major focus group for their future mobile platform. [9]

With this information in mind, it seemed important to understand how these popular social media platforms work in terms of their recommendation systems. The purpose of doing so could assist in identifying current trends and practices, while also understanding user expectations and preferences as these social media apps are popular and generally well-received.

2.1.1 Twitter

In a Twitter blog post, the three main stages of their recommendation pipeline are described. [56] In the first stage, a machine learning model most likely scoring high on the accuracy of a large number of recommendations is used to find candidate items (posts). Secondly, a different model presumably one that performs better in top recommendations is used to rank the candidate items. Thirdly, heuristics and filters are applied to remove items from sources the user has blocked or other items which might be considered inappropriate by the user.

Half of the initial candidates originate from sources the user is familiar with (e.g. follows or has interacted with), while the other 50% are from sources unfamiliar to the user. These items are also embedded and grouped into similar communities to determine which items and users are similar to the current user. Although the technology itself is not explicitly described, grouping users into "communities" strongly suggests a collaborative approach is at play. Most likely, it is a hybrid approach. Different types of approaches are described in greater detail later in this chapter.

2.1.2 TikTok

Developers have also made a similar blog post at TikTok. [23] From this, it seems their pipeline is similar to the aforementioned one on Twitter. However, after ranking the items which are deemed appropriate to show up on a user's feed,

they are checked for similarity. If two items are too similar, which could be if they use similar audio or imagery, they are replaced with a new item to ensure a greater variety in the content the user sees. To achieve this, the algorithm must take content properties into account, which implies the use of content-based filtering.

Examining these social media apps helps develop strategies to improve user retention and satisfaction with the new mobile application for Ekstra Bladet. This enhanced user engagement could be achieved by taking inspiration from this hybrid approach, which combines familiarity and exploration while ensuring content variety.

2.2 Recommender systems in news

Personalised content is widely applied in entertainment because increased engagement longevity and retention mean more clicks and time spent on an application or website. Similarly, news sites aim to achieve these goals, so applying the same technology is theoretically beneficial despite the change in context.

The New York Times published an article in 2021 which describes their, at the time, newly integrated personalisation algorithm. [53] Although a detailed explanation of their pipeline is not provided, which was the case with the previously mentioned blog posts by Twitter and TikTok, [2.1] their main concerns regarding implementation are documented.

The New York Times describes their approach as *"using recommendation algorithms to highlight articles that are particularly relevant to our readers"*. [53] This is done with a contextual approach, [9] as they consider a reader's geographical region, or reading history when judging which article would apply to a reader's interests.

Since the initial implementation of their recommendation algorithm, it has been expanded to include more contextual information according to the documentation. [53] The reader's device type, the time of day, and the number of articles viewed in a particular news section are all used to predict their interest in an article. An important note is that The New York Times found that the contextual information needed depends on the type of article that is being recommended.

The implementation itself requires two types of data: the articles read by the user and the articles shown to the user. The model is re-trained on the most recent version of this data and re-deployed every fifteen minutes. Additionally, the time frame of this contextual information is from the past 30 days. Although not explicitly stated, retraining at such frequency most likely also requires a robust monitoring system and a dedicated team, similar to Ekstra Bladet's data science team.

It should be noted that there is not a lot of publicly available information on major news sites using machine learning-powered recommendation systems. The likely reason is that already documented problems created by providing personalised content such as filter bubbles [8] could have a destructive effect on a brand's image, given that a common quality to strive for as a well-respected news publication is providing impartial and societally relevant information to the greater public. The New York Times' context-based filtering does not rely on "communities" nor limits what information a user is presented with based on the properties of their previously read articles. Thus, it can be considered a safe choice for avoiding the common dangerous side effects of recommenders - although most likely not as effective as collaborative and/or content-based approaches.

2.3 Recommender Systems

Examining some real-life applications of recommender systems, it became clear that based on the context of the system, developers must make some pivotal choices regarding the type of recommender system that is implemented. Other than exploring state-of-the-art standards, the development team must also establish a proper understanding of the different types of models.

Firstly, before exploring the intricacies of recommender systems, an essential understanding of what a recommender system does should be established. In the context of machine learning, a recommender system learns from data to predict what a user is looking for among item options, even in instances where the amount of options is exponentially growing. [34]

The criteria on which these predictions are based could, for example, be past purchases, search history, demographic information, etc. Many algorithms and techniques can be used to achieve this goal, but they are normally classified into three main categories; content-based filtering, collaborative filtering, and hybrid systems. These categories are explored to find the appropriate one for the issue at hand.

2.3.1 Content-based filtering

In content-based filtering, the predictions made by the recommender system are based on the properties of items a user has interacted with. In the case of articles, these could be topics, named entities, sentiment scores, etc. By including attributes and keywords associated with the articles, the model can create better predictions on similar content. [36]

In such a system, user profiles contain the user's interactions with the items

in the database. Attributes/keywords that appear in multiple interactions are weighted higher, as the system now perceives these as more important because they align more with the user's interests.

Content-based filtering has its advantages, such as being independent of other users' data. This is beneficial when working with limited user data, or when working in niche markets. Additionally, it is also highly personal, as it focuses on the individual user and their interactions with the items in the database. [36]

However, content-based filtering also has some disadvantages. For example, when recommendation systems end up being too focused on the user's past behaviour, their experience can be negatively impacted. The emphasis on past behaviour should be balanced with the rate at which new items are introduced. Otherwise, the system will be overly focused on previous trends while the user tries to engage with new content. The onboarding phase is also important to perform correctly when implementing content-based filtering, as it is difficult to gather the user's initial preferences otherwise - resulting in a cold start problem. A cold start problem arises when the system lacks sufficient information to make inferences about new users or items. [61]

2.3.2 Collaborative filtering

Collaborative filtering addresses some of the limitations of content-based filtering, more specifically, it addresses the issue regarding insufficient amounts of user interactions during the early stages of the user experience. This is tackled using serendipitous recommendations, meaning that the system recommends an item to one user, based on the interests of another similar user. Collaborative filtering uses similarities between users and items simultaneously to provide recommendations, which comes with its own set of advantages and disadvantages. [10]

An important positive aspect of using collaborative filtering is the fact that there is no need for metadata on the items within the database. This is because collaborative filtering only focuses on the interactions between the users and items. Additionally, it also mitigates the aforementioned issue regarding the onboarding phase in content-based filtering. This is because the system can still provide recommendations for new users, even when there is limited historical data available as it primarily creates the recommendation based on the behaviour of other users.

A disadvantage of using collaborative filtering is the risk of sparsity, which can occur if there are many users and items in your system, but without interactions between them. In such a case, the recommendations may risk being inadequate. Finally, a collaborative filtering recommendation system also tends to be biased toward popular items as other users gather toward those naturally, influencing other users in the same direction. [10]

2.3.3 Hybrid approach

Both content-based filtering and collaborative filtering offer unique advantages and face distinct challenges. While content-based filtering excels in personalisation and independence from other users' data, collaborative filtering leverages user interactions for serendipitous recommendations and mitigates the onboarding phase challenges. Hybrid recommender systems combine the strengths of both content-based filtering and collaborative filtering to mitigate their respective limitations and enhance overall recommendation quality.

There are several ways to implement a hybrid approach - the most common ones are described in this paragraph. The system can use a weighted approach where the content-based scores and collaborative filtering scores are weighted and aggregated to produce final recommendations, a cascading approach where

one recommender system refines the output of another, or a switching approach where the system switches between different algorithms based on contextual variables.

Based on evaluation results in scientific papers [6] [3], content-based filtering and hybrid approaches give the most outstanding results. The most optimal solution in the context of this project is likely a hybrid approach, but the possibility of such a system largely depends on the available data and the resources allocated for working on the model.

2.4 Deployment of recommender systems - Model serving

Once a model has been trained tested, and is considered ready to enter production, it means the model is ready to be served. Rather than a ship-and-forget pattern, machine learning systems require continuous improvement through iteration loops. A proper MLOps setup enables model lifecycle management and improves collaboration on an organisational level. [58]

The simplest way of serving a model is with model embedding, meaning that the model is stored and used on the device itself. It is generally regarded as a bad practice because it comes with massive client-side performance loss, any changes to the model require client-side updates, and it is not a scalable design. [58]

The prevailing approach currently involves deploying models through model-serving APIs or offering them as a service. This architecture divides the application from the model using an API, streamlining model version control and facilitating seamless updates through phased rollout processes, which do not directly affect users. To keep such a system loosely coupled, it is essential to keep

the application, database, model training, and model serving layers separated and modular. When serving a model, it is a good practice to enable continuous performance monitoring and easy switching between models. Implementing a model registry is also a good practice. A model registry is in simple terms a library of models stored independently from the API. [58]

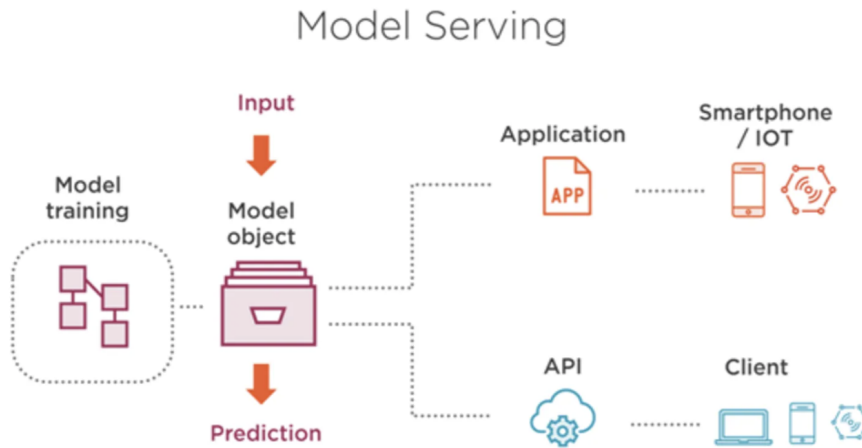


Figure 2.1: A general example for a model serving setup [58]

State-of-the-art model serving is often achieved using common API solutions like FastAPI [46] or Flask [47]. As an alternative, there are all-in-one solutions with user-friendly interfaces available out of the box. MLFlow [51] and Kubeflow [48] are examples of such tools, designed and priced for larger organisations.

2.5 Model evaluation

To find out whether a model performs well or not, and to compare vastly different models easily, models must be evaluated using universal metrics. As with most machine learning models, the first step in model evaluation is to define a training set and test set. The training set is a subset of data used to train the machine learning model. The model learns patterns, user preferences, item

characteristics, and interactions from this data. The test set is a subset used to evaluate the trained model's performance. It serves as unseen data, enabling an assessment of how well the model generalises to new, unseen instances. Typically, the data is split into training and test sets using an 80/20 - 70/30 ratio. This means that 70-80% of the data is reserved for training and the remaining 20-30% is used for testing. [4]

It is generally a good practice to use cross-validation, e.g. k-fold cross-validation, to ensure the model's performance is robust and not overly dependent on a particular train/test split. It is very important to avoid data leakage, which can occur if the splitting process is faulty and the model is trained on test data. Data leakages can lead to overly optimistic performance estimates. [40]

Once a model is trained on the training set, an appropriate evaluation metric should be used with the test set to provide meaningful evaluation metrics.

AUC is the industry standard evaluation metric for machine-learning-powered binary classification recommender systems. [21] AUC stands for Area Under the (ROC) Curve. The ROC curve is a plot of the true positive rate (sensitivity) against the false positive rate (1-specificity - the probability that a true negative will test positive) for different classification threshold values. The threshold value is a decision point that separates these classes - click or no-click in the context of this project. If the output of the model (e.g., predicted probability or predicted score) for a given example is above the threshold, the example is classified as positive; otherwise, it's classified as negative. AUC quantifies the overall performance of the model across all possible classification thresholds and represents the entire two-dimensional area underneath the entire ROC curve. A perfect classifier would have an AUC of 1.0, while a random classifier would have an AUC of 0.5 meaning it performs no better than chance. [21]

Providing several evaluation metrics is always a good idea. The F1 score is also commonly used as a secondary evaluation metric to account for false positives and false negatives (it is calculated using both precision and recall). Additionally NDCG (Normalised discounted cumulative gain - commonly NDCG@5 or NDCG@10), provides a score between 0 and 1 depending on the quality of the top recommended items. Although NDCG is most commonly used with search engines, it is still a popular evaluation metric in the domain of recommender systems. [15]

Chapter 3

Analysis

Having explored real-life implementations and established the central components of recommender systems, the development team is now ready to make educated choices and document the requirements for the most essential parts of the system, taking the context of Ekstra Bladet's situation into account. This section culminates in an overarching problem statement used as a guiding principle for what the upcoming work should contribute to providing a solution for, and big-picture requirements for the end product.

3.1 Context - Ekstra Bladet's data and current recommender system for web

Although the development team has not yet been authorised to access Ekstra Bladet's data, the information gathered during initial meetings strongly suggests there is a good amount of article and behaviour data to work with. [9](#) Knowing this, the development team is not hindered in taking either a content-based filtering approach, or a collaborative-filtering approach, and even possibly scaling such a model into a hybrid.

It has now been confirmed that the development team will not gain access to Ekstra Bladet's current recommender model or learn the specifics of its implementation. However, it has been documented that the model is used exclusively on the web platform to select relevant articles for a carousel view beneath an already clicked article. This information already opens up a lot of possibilities as to how user experience could be enhanced using a recommender on an entire application, rather than on a small subsection of the product. Therefore, the main focus lies in providing relevant content for a "for-you"-page, or in other words, the main feed the user sees when opening the application. When working with such a feed, both content-based and collaborative approaches are reasonable, and providing an entirely new personalised feed would, arguably, give the most value to the product - implementing machine learning where it matters the most.

Taking the RecSys Challenge [12] into account, with more and more powerful recommenders most likely emerging during the project's lifetime, although the development team's choice of recommender technology must be reasonable, providing an improved overall system should receive more focus than maximising model performance.

Based on research and the approximate mapping 1 of Ekstra Bladet's current system, it is clear that the recommender system requires its parts to be independent of, or at least only loosely coupled to the application. Recommender systems are computationally intensive, especially during training and evaluation. Furthermore, when a model is successfully trained and put into action, it should be usable with both the web and mobile platforms. Thus, it is most logical to keep model training and serving functionalities as separate components independent from each other. Lastly, the web and mobile platforms should use the same database for user interaction and article data, so it should receive its independent component as well.

Having established that the system must be highly modular, requiring largely independent components, the development team is now ready to explore the specifics of how such a system can be implemented in practice.

3.2 Course of action regarding MLOps

Building on the findings from the previous section, attention is now turned to the operational aspects of managing machine learning models, commonly referred to as MLOps. Simply put, MLOps focuses on streamlining the end-to-end process of deploying, monitoring, and maintaining machine learning models in production environments. [45]

In the context of Ekstra Bladet’s recommender system, MLOps is crucial for ensuring that the model training and serving components are scalable and easily modifiable or replaceable (more details to follow). The primary purpose of the model training component is to continuously improve the recommendation algorithms by leveraging new and improved models, while the serving component is responsible for delivering real-time recommendations to users across web and mobile platforms.

Given that model training is likely to be highly specific to the chosen model architecture, the model serving layer will receive more attention in terms of modularity and scalability. This should ensure that the serving layer can efficiently handle diverse and potentially evolving recommendation models.

Regarding state-of-the-art model serving solutions, experiments with MLFlow and Kubernetes using example models trained on the MIND dataset [7] (see appendix 9 for selection rationale) quickly revealed that the free tiers of these tools

lack many essential features required for production use. Therefore, model serving is planned to be done by hosting an API and eventually creating a Docker image and hosting that if possible. As formerly described, API solutions are common and provide a good starting point for multiple components to communicate with each other.

Tackling the issue of constantly receiving new articles and interaction data means that the API should have functionalities for partial retraining and loading new data from a remote database. Furthermore, it should be relatively easy to switch models out and evaluate models in production to monitor performance.

It is worth noting, that the importance of maintaining the model performance requires user interaction tracking - a feature not directly related to MLOps, but a must-have element of the product.

3.3 Course of action regarding recommender implementation

Now that the model-serving part of the system has been outlined, the development team must decide on exactly what machine learning model would make sense to work with. The consensus is that a better-performing model is naturally a preferable option, but the first and foremost priority is to have something that works, which can be changed out for better options in the future - potentially the winner model of the RecSys 2024 challenge [12].

The development team is faced with an initial choice between content-based filtering and collaborative filtering approaches - as for now, hybrid approaches are considered a possible future enhancement because of the complexities they come with. 2

Based on performance metrics with the MIND dataset [7], one most likely very similar to Ekstra Bladet's dataset, priority is given to well-performing off-the-shelf content-based filtering options like DKN, LSTUR, and NRMS. [6] They are a good fit for the issue at hand and are expected to provide the best results without any complex modifications. While selecting one of the aforementioned off-the-shelf models has several advantages, for instance, that they have been proven to perform well with similar data, scaling these models would require a thorough understanding of the inner workings of the chosen model. Additionally, most publicly available implementations do not consider using any of these models in a live production setting, which could propose very hard implementation tasks for the developers.

As an alternative option, if content-based filtering models are not production-ready or prove too difficult to tackle, there are numerous off-the-shelf collaborative filtering-based models available, with implementations ranging to production environments. Based on the documentation, performance, and available educational resources, LightFM [50] could be a very good candidate for this purpose (more about it later [4]).

To summarise, a better-performing model (content-based) is certainly preferred, but the point of this project is to provide a complete software system, not the perfect machine-learning model. If value delivery is greatly reduced by implementation difficulties, the development team should be quick to adapt and switch to a simpler approach - likely a worse-performing off-the-shelf solution.

3.4 Problem statement

The analysis and decisions outlined above lead to the core challenge the development team must address encapsulated in the following problem statement:

Enhance the user experience and improve user engagement on Ekstra Bladet's mobile application by developing a highly modular software system that supports a machine-learning-based recommender model. Provide a complete system and display a personalised news feed to users on an overhauled mobile application. Prove the success of the machine learning with an adequate performance evaluation process using relevant metrics.

3.5 Requirements

The problem statement defines the goal of the project, guiding the development team toward developing a comprehensive recommender system. To achieve this, the essential requirements are outlined encapsulating the tasks necessary for building the system.

The solution must be a complete system including independent components for model training, model serving, database, and application. It has been established that machine learning is a continuously developing field, and given that the current MLOps and database setup of Ekstra Bladet has not been disclosed, the plug-and-play modularity aspect should always be kept in mind when developing the solution. The focus is not as much on generating the perfect recommender model, but on creating a state-of-the-art platform for recommender systems to be used with.

The must-have components of a recommender system described above are en-

tered into the product backlog. Note, that these PBIs are subject to be broken down into smaller PBIs to ensure that tasks are of manageable size. PBIs are further specified once the technical stack has been established (see next chapter 4).

Considering frontend requirements, Ekstra Bladet has provided a Figma file to use as a design guide. 9 From this design, an initial list of PBIs is added to the product backlog as well.

Chapter 4

Technology stack

With a clear understanding of the problem statement and the essential requirements for a state-of-the-art recommender system, the next logical step is to explore the specifics of how these components will be built. The following section details the technology stack chosen to implement the proposed system.

4.1 Development environment

GitHub will be used for product backlog management, version control, issues, and continuous integration, as it is a state-of-the-art solution that all development team members are experienced with. GitHub's projects feature allows for setting sizes and priority labels on PBIs which are grouped into backlog, ready, in-progress, in-review, and done columns. Product backlog items are also linked to pull requests. GitHub actions are implemented for running automatic tests before merges such as linting. Any dependencies and bugs are documented as issues. During development, feature branching will be used, and each feature branch is closed upon merging into the development branch. At the end of each sprint, the development branch is merged into the master/main branch. For each pull request, a template should be filled out containing a description, screenshots (if relevant), corresponding backend or frontend branch, changes, related issues,

and a checklist.

The quality control checklist is the following:

- Code has been tested locally and passes all relevant tests.
- Documentation has been updated to reflect the changes, if applicable.
- Code follows the established coding style and guidelines of the project.
- All new and existing tests related to the changes have passed. (If one test fails because the test is broken, the function works as intended. The test has been ignored.)
- Any necessary dependencies or new packages have been properly documented.
- Pull request title and description are clear and descriptive.
- Reviewers have been assigned to the pull request.
- Performance impact of the changes has been evaluated, if relevant.

For setting up the development environments in both frontend and backend, readme files are written and continuously updated in the repositories.

4.2 Application Technology

For app development, React Native, NodeJS and Expo are used, because they are all well-maintained and documented, have lots of powerful libraries, and provide a headstart in getting an application to work on both IOS and Android.

4.3 Recommender Technology

Jupyter Notebooks and Google Colab will be used for model development and training. When working on a local Jupyter notebook, Conda will be used as an

environment manager and a package manager. Working with Conda simplifies installing and managing a wide range of Python and non-Python packages, libraries, and dependencies. The development team had previously experienced environment-related issues, so starting with a clean sheet and a good package management system makes sense.

For data preparation, sanitisation and manipulation, the Pandas library for Python will be used, as it is a powerful well-maintained tool all developers are familiar with. Pandas is generally good for working with tabular data in analysis and has many useful operations for data preparation and sanitisation.

FastAPI will be the development team's go-to tool for hosting an API for model serving. This is an option that was already discovered to be popular for this purpose during research. [2.4](#) Furthermore, the developers have already worked with FastAPI, which means a small headstart as well. For live-testing the API, Postman will be used, which is also a standard option the developers are familiar with. From an organisational perspective, MLFlow could be a better option, but as previously described [2](#) implementing it would both be financially costly and the rest of the system would be forced to be designed around it.

4.4 Potential off-the-shelf recommenders

As mentioned before, in the context of this project it makes most sense to go for a well-performing off-the-shelf recommender. Different types of recommenders, namely content-based collaborative, and hybrid approaches were previously described. This section documents the results of selecting the most suitable ones for this project.

To know what opportunities there are for off-the-shelf recommenders, and what

the most favourable choices would be, some of the most promising content-based filtering models and an easy-to-implement collaborative model were researched and documented.

4.4.1 DKN

The go-to off-the-shelf content-based filtering model of this project is DKN - Deep Knowledge-Aware Network for News Recommendation. This is the first choice because of its performance compared to similar content-based news filtering recommender systems documented in Wang et al.'s paper. [6] Furthermore, DKN is a content-based deep model specifically made for CTR (click-through-rate) predictions - exactly what is needed in this project.

DKN is a deep learning model incorporating data from knowledge graphs for recommendation purposes. [52] A knowledge graph is a data model that is used to explore relationships and identify logical connections in multiple datasets.

Observing performances and implementation, other potential candidates for content-based filtering could be LSTUR (Long- and Short-term User Representations) and NAML (Neural News Recommendation with Attentive Multi-View Learning). In short, LSTUR captures users' both long-term and short-term preferences and uses recurrent neural networks. NAML uses an attention mechanism to dynamically weigh the importance of different views (e.g., user features, news article features) during the recommendation process. Both of these options give promising results with the MIND dataset. [6] These recommenders are not explored further for now, but are kept as possible alternatives if implementing DKN in the system is not feasible.

4.4.2 LightFM

LightFM has been determined to be the safest choice for an off-the-shelf recommender model. Other than proper documentation, maintenance, and solid community support, LightFM performs relatively well compared to other collaborative filtering and even content-based options. [6]

LightFM is a CPU-based Python implementation of a Factorisation Machine recommendation algorithm. [49] LightFM does not currently have a GPU-based implementation, so training times are expected to be slower than other potential options. [50] LightFM being a factorisation machine model means that it is a supervised learning algorithm (works with labelled training data) designed to capture interactions between features within high dimensional sparse datasets. Factorisation machines are most commonly used for classification problems, but can also be used to tackle regression tasks. Click prediction systems are a prime example of factorisation machine models for binary classification. [43]

LightFM can be used as an interface to handle both collaborative and content-based filtering. Although the most popular use of LightFM is with the WARP loss function, which is a popular choice for collaborative filtering tasks. Simply put, WARP is used to optimise recommendations based on the relative ranking of items. WARP is especially useful when working with binary input describing whether or not an interaction has taken place - which also makes sense for this project. Due to the vast amount of available resources for WARP implementation and the impression that it fits well with the context of the project, this seems to be the best option for a fast workflow.

In LightFM, users and items are represented as linear combinations of latent factors of their content features. Latent factors are a transformation of the data

points to "explain" patterns in the observed data. In recommendation systems, latent factors can represent the underlying characteristics of users and items.

These representations produce scores for every item for a given user, and an item receiving a higher score means that the item is likely to be more relevant to the user. A representation for each user/item combination is a linear weighted sum of its feature vectors. LightFM operates based on binary feedback, so feedback is normalised into two groups (click/no-click in this case). Since LightFM is constructed to predict binary outcomes, it uses a sigmoid function. [49]

LightFM also has some specific constraints compared to machine learning standards. LightFM expects training and testing sets to have the same dimension, therefore conventional test split will not work. To tackle this, LightFM has its unique method for data splitting. Note that this method does not guarantee that all user-item interactions in the test set have historical interactions in the training set. [49]

4.5 Database

Supabase will be the developers' choice for cloud data storage. Its free tier will not propose any severe limitations for the developers, and relations between tables can be defined, unlike in MongoDB. Relational databases are preferred when working with structured data. The data received from Ekstra Bladet is already structured so it is good practice to keep using their structure and type of database.

Functions for user behaviour instrumentation will be written by the developers. Amplitude [**aplitude**] was explored as a possible off-the-shelf option, but free tiers proposed too many limitations and front-end design constraints.

4.6 System architecture

The product in this report is built with a layered system architecture in mind. In a layered system architecture, layers handle different operational aspects of the system. There are various sub-modules inside the layers to aid the layer's purpose in the system. [2]

The primary reason behind choosing layered architecture is its modularity. The modularity with sufficiently loose coupling brings improved independence between the different components, which is beneficial since system components are meant to be easy to replace or modify without affecting other areas.

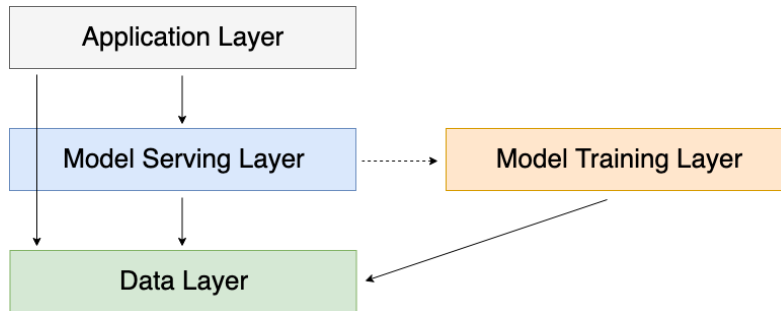


Figure 4.1: The system's layers where the arrows indicate requests

The structure of the system's layers is shown in the figure 4.1. The layers are; application, model serving, model training and data layer. Usually, the communication, via requests, between the various layers is hierarchical and a layer can only depend on its neighbour. The reason for this is to gain the full effect of the flexibility and modularity that layered system architecture provides. When there are fewer dependencies between each layer, a system presents a better foundation to replace and change layers if needed. In this project, the communication between layers is more dependent than only relying on their closest neighbour. The reason is that the application layer can both send a request to the model serving layer and data layer depending on what type of request it is sending. [27] [57]

Chapter 5

Implementation

In previous chapters, the problem was thoroughly researched, general requirements for a potential solution were outlined, and those requirements were translated into specific technologies. With a clear plan in place and a firm understanding of the project's overarching goals, the actual development of the product can now begin.

In this chapter, a detailed overview of the system architecture and a brief run-down of Ekstra Bladet's EBNeRD dataset is first presented. The Agile workflow is described followed by a documentation of all main developments during sprints. The report highlights the most important items addressed in each sprint, while the remaining items are included in the appendix. The purpose of presenting the system architecture before the implementation process is to provide context for the documented developments.

5.1 System architecture in detail

In the previous chapter, the type of system architecture used in this project was explained. [4.6](#) Now, the exact details of how the system architecture is implemented are presented. A diagram of the whole system is displayed in figure [5.1](#)

to get a closer look at the different components placed in the layers.

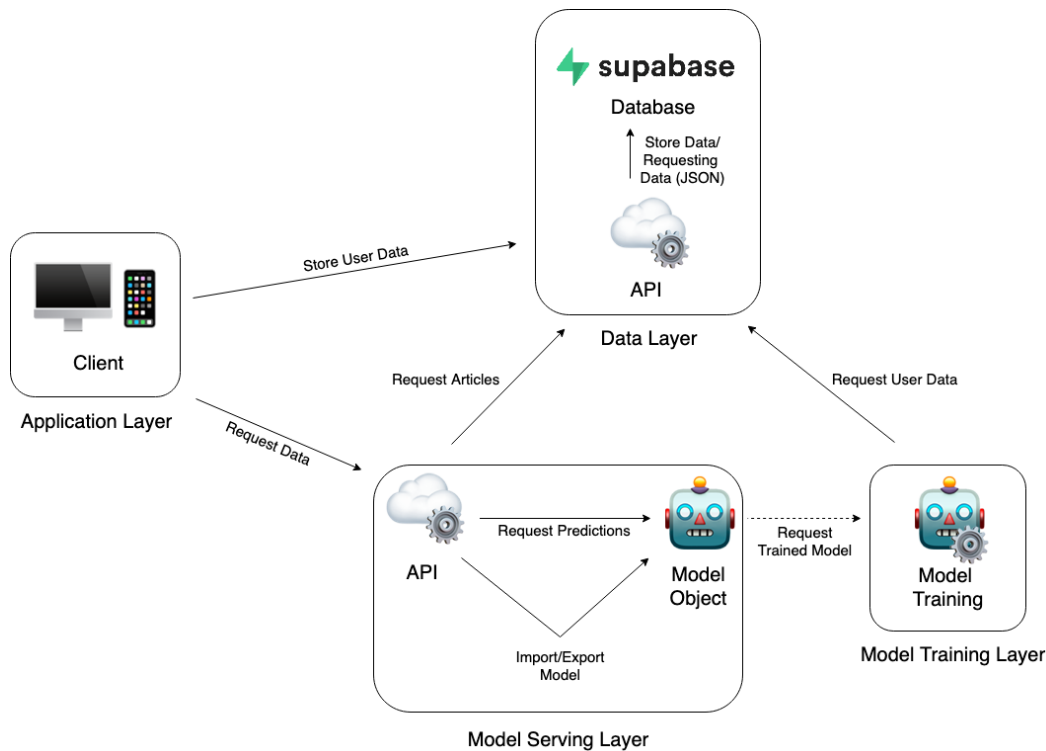


Figure 5.1: System architecture

Application Layer

The client in the application layer currently represents a mobile device, but can easily be extended to support web display. This layer starts the process of communication between the layers by either requesting data from the model serving layer - which contains the model object used for providing recommended articles - or storing user interaction data in the data layer. Both are done by the use of API requests and responses. The developers' self-defined FastAPI instance is used in the model serving layer and Supabase's provided API is used to communicate with the database in the data layer.

Normally, the client only handles the UI and is often placed inside a presenta-

tion layer. In this case, the application and presentation layers have been merged since a fair amount of business logic is stored on the client side. Therefore, the client and its business logic represent the application layer in this context. This approach limits the modularity since it will take a lot of effort to change the client and reallocate the business logic elsewhere. The same limitations apply to the dependency on the data layer.

Model Serving Layer & Model Training Layer

Model serving and training layers are on the same hierarchy in the system. This is because the model serving layer can send a request to either the data layer or the model training layer independently.

The model serving layer, via an API, handles the requests from the application layer to get predictions from the trained machine learning model object. Furthermore, the API handles the import and export of model objects to be used for predictions and, lastly, the request for articles matching the article IDs obtained by the model's predictions from the data layer.

The arrow from the model serving layer to the model training layer is not solid, because the current system setup does not perform a request as intended. Getting a model into production is a manual process with the use of a Google Colab Notebook, where the model is trained and evaluated, and from which the trained model is exported as a joblib file. This file is manually inserted into the model folder in the model serving project locally. The reason for this is that the LightFM model imported from the recommender's GitHub repository uses some tools that are readily available on Linux, but not on Windows or Mac. Workarounds were attempted but they always resulted in kernel failures, so to avoid slowing development velocity, the model training work was decided to be kept on Google Colab.

Ideally, the model serving layer could send a request to trigger the training of a new model in the model training layer. The model training layer would then request the data layer to get the latest user interaction data. After the training is done, the newly trained model would be returned to the model serving layer or put into a model registry. Here it should be up to the developer to choose if they want to utilise the most stable and highest-performance model available in production or observe more performance metrics before switching the models out.

Data Layer

The data layer consists exclusively of elements made by external parties. It includes a cloud relational database which has an API. The API can send a POST or GET request to either place data inside one of the database's tables or get data from a table in JSON format.

The request and response data formats, respectively SQL and JSON, are widely used when working with databases. Therefore, finding an alternative database provider with the same format might be fast, and can result in spending a relatively small amount of time to replace the existing dependencies to integrate with it. [62] [16]

5.2 The Ekstra Bladet News Recommendation Dataset (EBNeRD)

The EBNeRD dataset is created by Ekstra Bladet to assist in the research in news recommendation research. [11] It is available in different sizes, namely demo, small and large. The small dataset is primarily used for this project, and the demo dataset is at times temporarily used for testing purposes. The small dataset

was chosen by comparing results and training times between all three sizes. Although the large dataset results in a slight increase in model performance, the tradeoff in training time is too large to make it a viable option.

The small dataset includes 20738 articles with textual content features such as titles, abstracts and bodies. The dataset also comprises 15143 users, and 232887 impression logs of user behaviours recorded on Ekstra Bladet's website. Users and their interactions are split up into behaviour and history data files, where the behaviour data consists of a list of individual interactions and their properties, and history encompasses a unique user total interactions into one row.

It is worth noting that during the first sprint, the RecSys Challenge was not yet made public, so it was uncertain whether the development team would be able to access this data. Therefore, similar publicly available datasets were also explored to train and validate a potential recommender model. ⁹ During the first sprint, the MIND (Microsoft News) dataset was temporarily used as a stand-in for EBNeRD.

5.3 Agile workflow and collaboration with Ekstra Bladet

The project workflow applies some, but not all, fundamental aspects of Agile software development. This decision was made because Agile practices provide a good framework for continuous value delivery to the project's stakeholders and keep the priorities and development steps on track working towards concrete goals.

The product owner of this project is Kristoffer Hartwig, who works as a product developer at Ekstra Bladet. Hartwig's primary responsibilities include keeping the product vision on track representing stakeholder needs, managing priorities

and assisting in defining product backlog items. Additionally, they have the final word in accepting product backlog items.

The management role in this project is fulfilled by Andres Masegosa, associate professor at the Department of Computer Science at Aalborg University, and the development team's supervisor. Masegosa's primary responsibilities include providing guidance and direction to the development team, thus ensuring continuous improvement and value delivery.

The development team consists of the authors of the report - Software students from Aalborg University Copenhagen. The development team's responsibility is to continuously deliver valuable increments while ensuring sufficient transparency and common understanding with other key figures. Considering that all team members are familiar with Agile principles, it was decided that a Scrum Master-like role is not required.

The development process consists of two-week sprints. Each sprint starts with a sprint planning session using product owner and supervisor input to solidify the main goal and product backlog items. During development, the development team holds daily meetings internally to provide progress updates. After the 2 weeks of development, all PBIs considered done - see Definition of Done below - are presented to and discussed with the product owner and supervisor in the form of a sprint review. During sprint reviews, feedback considering the overarching goal and concrete PBIs for the next sprint is collected and systematically documented. If needed, a sprint retrospective is held discussing any issues with the development process or events.

The definition of done is defined as the following:

1. Code has been tested locally and passes all relevant tests.
2. Documentation has been updated to reflect the changes, if applicable.
3. Code follows the established coding style and guidelines of the project.
4. All new and existing tests related to the changes have passed.
5. Any necessary dependencies or new packages have been properly documented.
6. Pull request title and description are clear and descriptive.
7. Reviewers have been assigned to the pull request.

The implementation of the product follows Scrum principles and is therefore documented in sprints. These are iterative development cycles which follow a specific structure, with the main goal being to implement the items from the product backlog while also adhering to the definition of the done. The items are defined in coordination with the product owner, Kristoffer Hartwig, and the project's supervisor, Andres Masegosa. It is crucial to be transparent and keep the product owner in the loop because they are responsible for the overall product vision, accepting PBIs, and providing meaningful feedback. The supervisor's expertise in machine learning is pivotal for making meaningful technical progress on the product with good velocity and keeping the project on track. The structure is maintained by including the key activities of a Scrum framework, these are Sprint Planning, Sprint Review and Sprint Retrospective.

The purpose of documenting these sprints is to track the progress of development and provide a reference for future iterations and improvements. Furthermore, each sprint has a clear overarching goal it works towards, defined in collaboration with the product owner's needs and supervisor's advice. Every new

contribution must therefore be made with this goal in mind.

For tracking PBIs, they receive their unique identifiers, formulated as a concatenation of their sprint number, and either the front or backend number assigned. For example, S1F2 is the second frontend PBI from sprint 1, and S2B3 is the third backend PBI from sprint 2.

5.4 Sprint 1

In this first sprint, the goal was to get all crucial elements of the system up and running. This entails creating a React Native application according to the Figma design provided by the design team at Ekstra Bladet and implementing a machine-learning model with the data provided by Ekstra Bladet. Combining these two fundamental elements of the system, the application should display content which is provided and recommended by an external machine learning model through API calls. Note, that the quality of the machine learning model (evaluated as AUC score), is not considered important as of now.

5.4.1 Sprint Planning

The planning phase of this sprint included a meeting with the product owner to ensure a good common understanding of the sprint goal. Currently, the product owner prioritises and has a better understanding of requirements considering the application frontend, thus the backend (primarily machine-learning related) requirements were formulated following the supervisor's guidance. For each PBI, a new branch and at last PRs are created and they are labelled either "In backlog", "Ready", "In progress", or "Done" corresponding to the progress.

The following is a list of the backlog items addressed during this first sprint. The items with the most significance, which are the ones highlighted in the list,

are expanded upon in this section. The rest of the completed PBIs from this sprint can be found in the appendix. [9](#)

Front-end items

- S1F1: As a user, I want to have a non-personalised news feed.
- **S1F2: As a user, I want to have a personalised news feed.**
- S1F3: As a user, I want a header on my personalised feed.
- **S1F4: As a user, I want to be able to read a news article in the app.**
- S1F5: As a user, I want to be able to see if I'm a plus member or not.
- S1F6: As a user, I want to navigate from the news feed to an article by tapping on it.
- **S1F7: As a user, I want to see more articles when refreshing the news feed or reaching the bottom of my news feed.**
- S1F8: As a user, I want to see a loading screen with Ekstra Bladet's logo when opening the application.
- S1F9: As a user, I want to see the correct fonts when using the app.
- S1F10: As a user, I want to have paragraphs in the article.
- **S1F11: As a user, I want to see image embeddings.**
- S1F12: As a user, I want to see breaking news with special styling.

Back-end items

- S1B1: Construct a recommender model and train it on the EBNeRD dataset.
- S1B2: Evaluate the model using AUC and F1 scores.

- S1B3: Make an API that returns a list of articles when requested for a prediction using the model.
- S1B4: Make the number of recommended news to a variable in the API call.

These items align to create a stable and functioning application with the implementation of the minimum features wanted by Ekstra Bladet, displaying Ekstra Bladet's article data recommended by a machine-learning model.

S1F2: As a user, I want to have a personalised news feed. Figure 5.2a

This PBI focuses on creating a component, which will become an essential part of the landing page of the application - displaying recommended content to the user. It was designed with the design provided by Ekstra Bladet in mind, which can be found in the appendix 9. This PBI touched on two files, the news feed screen, and news card components that populate said screen. The news card component contains the following:

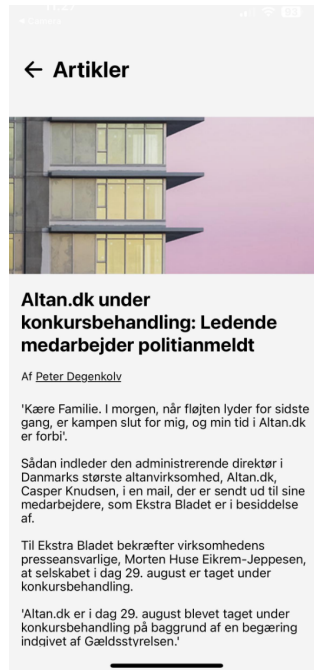
- Small picture of journalist
- The journalist's name
- The category of which the article belongs in
- How old the article is
- A thumbnail for the article
- The title of the article

Additional news cards render as the user scrolls, initially using mock data since the machine-learning model and frontend were developed concurrently. Thumbnails require image embeddings from Ekstra Bladet's image database, managed by a separate PBI, which poses a challenge as the images are not stored in the

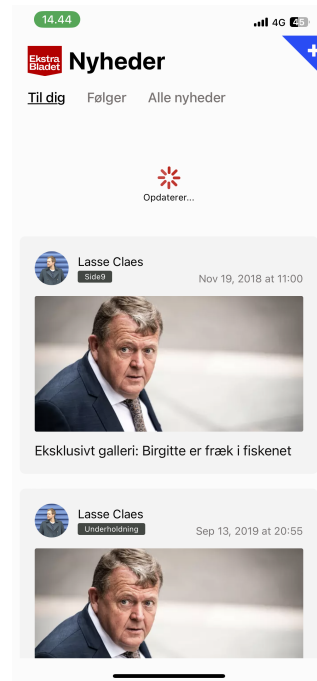
development team's database. While each card is touchable, they do not yet respond to presses because this feature is part of a separate PBI.



(a) news card component



(b) Article screen



(c) refresh component

S1F4: As a user, I want to be able to read a news article in the app. Figure 5.2b

This PBI is related to the news card component and addresses the screen displayed when a user presses a news card to read the full article. The fonts for this screen are not included in this PBI and are handled later in the sprint. Additionally, the functionality to navigate from the newsfeed screen to the article screen upon pressing a news card is also addressed in a later PBI.

S1F7: As a user, I want to see more articles when refreshing the news feed or reaching the bottom of my news feed. Figure 5.2c

This PBI addresses the functionality of refreshing the news feed, while also displaying more articles when reaching the bottom of the initially rendered articles.

This design was not a part of the designs created by Ekstra Bladet and was therefore created by the development team and accepted by the product owner in a following meeting. The backend provides a specified number of recommendations to the user, and when new ones are needed during refreshing or reaching the end of the feed by scrolling, the articles shown are from the same pool as the model is not providing new recommendations. There is a known and documented issue - the loaded articles are the same as the ones already present on the feed. This issue stems from the backend always taking the number of recommendations from the top of the ranked predicted articles, rather than providing different ones. This issue should be addressed as soon as possible.

S1F11: As a user I want to see image embeddings

This PBI enhances the news card component, as well as the articles themselves, providing correct images corresponding to article IDs. Tackling this PBI, an attempt was made to reverse engineer how image links are constructed by finding the articles from the dataset on the actual website and inspecting the links. It was swiftly discovered that the links follow this structure:

`https://img-cdn-p.ekstrabladet.dk/image/ekstrabladet/image_id/
relationBig_910/?at=some_hashed_number`

To find out what hashing function is used, the development team contacted the product owner, who assisted the development team by getting in contact with an internal developer at Ekstra Bladet. It was revealed that the final part of the link is a hex digest given by MD5-hashing the following hash string:

`articleId-publishedTimestamp-modifiedTimestamp`

This was then swiftly implemented in a backend function that can be run to append an *image_url* column to items that are passed to the app with an API call:

Listing 5.1: get article URLs code

```
1 def generate_image_url(self, image_id, article_id, published_timestamp,
    modified_timestamp):
2     hash_string = f"caravaggio-{article_id}-{published_timestamp}-{
        modified_timestamp}"
3     md5_hash = hashlib.md5(hash_string.encode()).hexdigest()
4     return f"https://img-cdn-p.ekstrabladet.dk/image/ekstrabladet/{image_id}/
        relationBig_910/?at={md5_hash}"
```

The *generate_img_url* function takes four parameters: *image_id*, *article_id*, *published_timestamp*, and *modified_timestamp*. It generates a hash string using the provided parameters, then it calculates the MD5 hash of the hash string. Finally, it constructs and returns a URL using the *image_id* and the calculated MD5 hash as parameters.

Now, when making an API call requesting recommendations, these functions are run on the recommended items before returning, so that the app can use these image links to create image embeddings. As a failsafe for any potentially missing images, if the image ID is not available, then a default placeholder is sent back with the response.

5.4.2 Sprint Review

In the meeting with the product owner, all frontend PBIs were accepted, with some minor design tweaks requested for font selection and layout, and the issue of loading new articles at the bottom of the feed was promptly resolved. Having already implemented a significant portion of the user interface, breaking it down into individual components is expected to provide a solid foundation for incorporating more advanced features in the app.

Regarding the recommender system, numerous attempts were made to implement promising content-based filtering options (DKN, LSTUR, NRMS), but none were successful. It was determined that these systems were not suitable for production. Diverging from the “intended path” and attempting to transfer any of these models into practical use for real-time predictions proved too difficult. As a quick replacement, a simple PyTorch-based collaborative filtering model was implemented for the EBNeRD data. Code was written to evaluate its performance (0.502 AUC, barely better than chance), and it was hosted using FastAPI to provide a presentable starting point for the product. Although the model was rudimentary and was merely a temporary solution, it enabled the development team to set up Axios requests on the frontend, eliminating the need for mock data.

The product owner and Ekstra Bladets data science team had no strong opinions on the recommender itself regarding the technology used and implementation methods. Still, they did request training the model with meaningful features - an intuitive next step in the process, preferably after pivoting to a better-performing off-the-shelf option.

5.5 Sprint 2

In the first sprint, the essential parts of the application, and its communication with an impromptu served model were established. Therefore, in sprint 2, it is now possible to begin the implementation of user interaction tracking, and improving the quality of the machine learning model. It is worth noting that the development team, as well as the product owner, took some days off to celebrate Easter - thus, communication was not as effective during this sprint, and the delivered value was reduced too.

5.5.1 Sprint Planning

The following is a list of the backlog items which are addressed during this sprint. The minor frontend fixes and adjustments requested by the product owner are not documented as PBIs. The items with the most significance, which are the ones highlighted in the list, are expanded upon in this section. The rest of the completed PBIs from this sprint can be found in the appendix. [9](#)

Front-end items

- S2F1: Change the font in the article screen to one that looks more readable (Sans-Serif).
- **S2F2: Track clicked articles, corresponding scroll percentages, and read times in the app.**

Back-end items

- S2B1: Include meaningful features in the machine learning model to improve its performance.

S2F2: Track clicked articles, corresponding scroll percentages, and read times in the app.

User behaviour tracking is the first step in consistently providing new data to the machine learning model to continuously improve personalised predictions. As a start, clicked articles, scroll percentages, and read times are tracked because these properties (other than the ones directly available from article IDs) were initially recommended by Ekstra Bladets data science team.

In the newsfeed screen, `clickedArticleIds` and `scrollPercentages` are state variables used to store the IDs of articles that the user has clicked on and the scroll percentages of those articles.

setUserHistory function retrieves the user's history, including clicked article IDs and scroll percentages, from the device storage asynchronously using getUserHistory, and updates the list of clicked articles and scroll percentages.

handleGoBackFromArticle function updates the clickedArticleIds and scrollPercentages states when the user navigates back from an article.

Inside the ScrollView component, for each article in the articles array, a NewsCard component is rendered. The scrollPercentage for each article is obtained from the scrollPercentages state based on the article's ID. This will be useful for a UI enhancement suggested by the product owner, indicating much of an article the user had previously read.

In the article screen, scrollPercentage, scrollHeight, and readTime are state variables used to track the scroll percentage, scroll height, and read time of the article, respectively.

The handleScroll function is triggered whenever the user scrolls within the article. It calculates the current scroll percentage based on the scroll position and updates the state accordingly.

Listing 5.2: track scroll percentage

```
1 const handleScroll = (event) => {
2   const { contentOffset, contentSize, layoutMeasurement } = event.nativeEvent;
3   const currentScrollHeight = contentSize.height - layoutMeasurement.height;
4   const currentScrollPercentage = (contentOffset.y / currentScrollHeight) * 100;
5   setScrollHeight(currentScrollHeight);
6   setScrollPercentage(currentScrollPercentage);
7 };
```

Read time is tracked using the `useEffect` hook. The read time variable is incremented every second.

Listing 5.3: track read time

```
1 useEffect(() => {
2     const interval = setInterval(() => {
3         setReadTime((prevReadTime) => prevReadTime + 1);
4     }, 1000);
5     return () => clearInterval(interval);
6 }, []);
```

The `handleOnBack` function is called when the user navigates back from the article. It saves the article's ID, timestamp, read time, and scroll percentage using `AddClickedArticle` and emits an event using `emitGoBackFromArticle` to update the user's history.

Inside the `ScrollView` component, the `onScroll` event is used to call the `handleScroll` function, updating the scroll percentage as the user scrolls through the article.

5.5.2 Sprint Review

In this sprint, the development team successfully implemented user behaviour tracking, an essential system component.

Regarding the machine learning model, although no substantial work was done, the previously described `LightFM` model was deemed fit for production. [4] This was determined by reading its documentation [50] and exploring publicly available implementations [49] of the model.

Additionally, Ekstra Bladet publicised the leaderboard for the `EBNeRD` recommender model competition. With several models scoring around 0.65-0.70 AUC,

the development team is advised to focus on providing a modular system with an off-the-shelf recommender that can be easily swapped for a better-performing model from the competition. This approach offers a more future-proof solution, as models will likely continue to improve over time, necessitating eventual replacement. In the next sprint, the group's model will need to be replaced while still fulfilling all previously completed model-related PBIs.

Specifically, this entails training and evaluating an off-the-shelf model on the EBNeRD dataset using meaningful features, visually representing the results to observe convergence, exporting the model, and writing API code to communicate with the application and return predictions.

To ensure a realistic solution, model serving is also relevant, and understanding Ekstra Bladet's current recommender setup (MLOps) is important. During the sprint, the development team consulted Ekstra Bladet's data science team to identify which model aspects (features, architecture, etc.) to focus on, gaining insight into the most critical properties. They also inquired about Ekstra Bladet's tech stack and model serving method (MLOps). However, they did not receive answers before the sprint ended due to limited availability during the Easter holiday.

With the focus now on providing a robust system, completing the entire pipeline is vital. So far, the application, model training, API calls using the exported model, and the fundamentals of behaviour tracking have been implemented. While the functionality is there, layers of the system must be more clearly separated, ensuring loose coupling and modularity. After replacing the model and rewriting the API code, the group must implement a database solution containing behaviour and article data, serve the model, and incrementally train the model on new data.

5.6 Sprint 3

The primary focus of this sprint is replacing the current model with a well-established off-the-shelf model, LightFM, while completing every previous model-related PBI with a complete system in mind, thus enhancing overall modularity and improving prediction performance. If possible, other minor enhancements on the frontend sides are also welcome, but all of these should be small items with quick implementation velocity.

5.6.1 Sprint Planning

The following is a list of the backlog items which are addressed during this sprint. The minor frontend fixes and adjustments requested by the product owner are not documented as PBIs. The items with the most significance, which are the ones highlighted in the list, are expanded upon in this section. The rest of the completed PBIs from this sprint can be found in the appendix.[9](#)

Front-end items

- S3F1: As a user, I want to see how much of an article I have previously read.

Back-end items

- **S3B1: Implement the LightFM recommender model and train it on the EBNeRD dataset.**
- **S3B2: Train the model using meaningful features.**
- **S3B3: Evaluate the model using the AUC score.**
- S3B4: Provide a visual representation for the training and validation AUC scores for each epoch to depict improvements and convergence.

- **S3B5: Make an API that returns a list of articles when requested for a prediction using the model. The number of articles returned must be a variable.**

S3B1: Implement the LightFM recommender model and train it on the EBN-eRD dataset.

In the recommender library example [49], the implementation of LightFM uses the MovieLens dataset. This is a good starting point for development because the tabular input data consists of a userID, itemID, rating, and genre. By performing some data preparation steps, Ekstra Bladet’s user history and behaviour data can be reformatted to look the same. Properties userID, itemID and genre (topic labels) are given, rating is switched out for 0 if an article is viewed but not clicked, and 1 if an article is viewed and clicked. Having properly formatted the data, the recommender library’s implementation was now also usable with the EBnERD dataset.

userID	itemID	rating	genre
123456	518008	1	Sport

Note, that every interaction requires its unique row, so quite a bit of work is necessary to reformat the original EBNeRD data. This process can be found in the appendix. 9

S3B2: Train the model using meaningful features.

It is accepted for now, that using the article category is a meaningful enough feature, but it is of course advisable to scale the model to include more features to improve its performance. Knowing that any decently performing would be acceptable, potential scaling is kept in mind but is not a high priority.

S3B3: Evaluate the model using the AUC score.

It was previously established that AUC is the primary evaluation metric for the recommender model (also in the RecSys Challenge), therefore, it is a priority to implement.

The implementation includes the previously included safety check ensuring that there is no overlap between training and validation data, although it is already checked during data preparation. Subsequently, the AUC score is calculated using LightFM's `auc_score` module, and the average AUC score is calculated at last.

Listing 5.4: calculating AUC score

```
1 # Exclude interactions in the training set from the test set
2 test_interactions_excl_train = test_interactions - train_interactions.multiply(
    test_interactions)
3
4 # Calculate AUC score
5 with Timer() as auc_time:
6     auc_scores = auc_score(model, test_interactions=test_interactions_excl_train,
7         num_threads=NO_THREADS)
8
9 # Get the number of AUC scores and calculate the average AUC score
10 num_auc_scores = len(auc_scores)
11 average_auc_score = np.mean(auc_scores)
```

This implementation was then modified to be incremental, which comes in handy in finding convergence. This code calculates the average AUC score for each epoch run and saves these scores so they can be plotted. Using the partial fit function also means training can continue by simply rerunning this cell multiple times.

Listing 5.5: incremental AUC score check

```
1 # Store AUC scores for each epoch
2 train_auc_scores_per_epoch = []
3 test_auc_scores_per_epoch = []
4
5 for epoch in range(NO_EPOCHS):
6     # Fit model for current epoch
7     model.fit_partial(interactions=train_interactions, epochs=1)
8
9     # Calculate AUC score for current epoch
10    train_auc_score_epoch = auc_score(model, train_interactions, num_threads=
        NO_THREADS)
11
12    test_auc_score_epoch = auc_score(model, test_interactions=
        test_interactions_excl_train, num_threads=NO_THREADS)
13
14    # Append AUC score to list
15    train_auc_scores_per_epoch.append(np.mean(train_auc_score_epoch))
16    test_auc_scores_per_epoch.append(np.mean(test_auc_score_epoch))
```

As for now, the model was only trained for 20 epochs, so no overfitting nor convergence can be seen on the plots. Still, the model's performance is already considerably higher than the temporary model's 0.502 AUC score. Still, by modifying the parameters, increasing the number of epochs to train on, and potentially including more relevant features, the AUC score could be much better than it is at the moment (0.53).

S3B5: Make an API that returns a list of articles when requested for a prediction using the model. The number of articles returned must be a variable.

When creating the API, it was important to keep in mind that if such a system was put into production, it would most likely be used on both the app and web interfaces of the news site. This means that any requests and responses should be independent of the platform. The API for making predictions using an exported

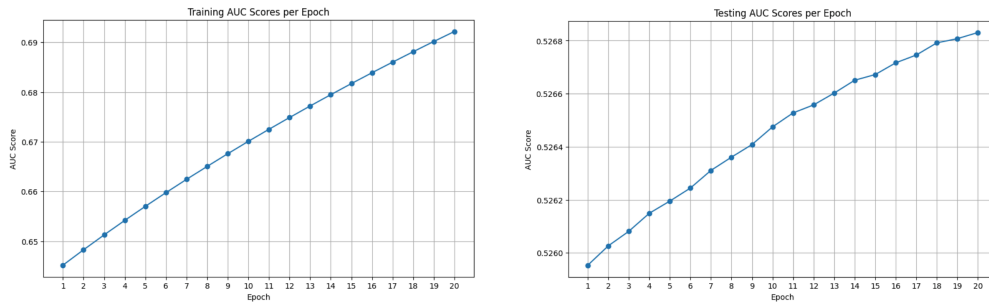


Figure 5.3: AUC scores of the LightFM model without any modifications after 20 epochs

machine-learning model is divided into three parts.

Firstly, the API itself contains pedantic models for requests and responses and endpoints with which an app or web application can make calls. These requests include getting all news in a non-personalised way, which requires a starting index and number of news articles to be passed. The starting index is used when refreshing the feed or loading new articles. When making predictions, a userID is also passed next with the aforementioned two properties. When starting the API, news and model data are all loaded, and the recommender system is instantiated (more on that in the next paragraph), so prediction times can stay fast, normally a fraction of a second. The API is hosted on a self-defined URL, which can easily be adapted to a physical or cloud-serving setup such as AWS or Google Cloud Services, depending on the needs of the owner.

The second component is the LightFM Utilities, where all of the logic for making predictions is placed. The class has a function to load a model, right now simply from an OS path, but this can be changed out to refer to a cloud storage solution very easily. In preparation for enabling model retrains in the future, user behaviour data is loaded when initialising the recommender system, and it is encapsulated in a function to enable data reloads when new data comes in. When loading the data, LightFM's internal indexing is also updated. Most im-

portantly, the LightFM utility component has a function that makes predictions for a user. In short, the passed user ID and all articles are internally mapped and passed to LightFM's prediction module. It is worth mentioning that in a production setting, you would only want to recommend the newest articles - luckily this parameter could easily be adjusted. The predict function only returns a list of scores for all articles, so some additional steps must be taken to convert this into a usable JSON response. In short, actual article IDs are retrieved by translating the LightFM internal IDs into real article IDs using a mapping function. Subsequently, the scores are sorted in a descending fashion while keeping their indices, so that the actual article IDs can also be rearranged the same way. Then, the requested number of top-scoring articles is extracted and selected from the article data. Finally, the article data is returned.

The third component is News Utilities. This file encapsulates the formerly described logic for generating image embedding URLs and retrieving the requested number of the latest news.

Throughout each of the components, several failsafes and validation steps are put in place to ensure that, for example, a given user ID exists, or that a generated response is correct.

5.6.2 Sprint Review

The main goal of the sprint was successfully fulfilled, with LightFM fully implemented in the system, maintaining or improving the previous implementation of several PBIs related to model serving. The need for the development team to switch the model proved to be a valuable test of the product's modularity. With the model now in place, the primary focus is to fill in the blanks and deliver a complete system to the product owner, while experimenting with different model configurations to increase its performance and provide better recommendations.

5.7 Sprint 4

The overarching goal of this sprint is to deliver a complete system. Specifically, this includes an overhauled app with all key features and behaviour tracking, data storage in a cloud service, and a model serving setup with an API that works for both the web and the app. The system should be flexible enough to retrain and switch models easily. Beyond delivering the system, it is crucial to support its plug-and-play capability as broadly as possible without imposing too many requirements.

5.7.1 Sprint Planning

The following is a list of the backlog items which are addressed during this sprint. The minor frontend fixes and adjustments requested by the product owner are not documented as PBIs. The items with the most significance, which are the ones highlighted in the list, are expanded upon in this section. The rest of the completed PBIs from this sprint can be found in the appendix. [9](#)

Front-end items

- **S4F1: As a user, I want to see more recommended articles under a clicked article to keep me engaged.**
- **S4F2: Store and continuously update user behaviour tracking data in a database. Save both as the format provided by Ekstra Bladet, and the format used in the model.**

Back-end items

- **S4B1: Implement model serving.**
- **S4B2: Enable developers to switch between different models without rewriting the API code.**

- **S4B3: Implement model retrain for a served model.**
- **S4B4: Implement performance monitoring for a served model.**
- S4B5: Improve performance monitoring by using TensorBoard.
- **S4B6: Adjust training parameters to improve model performance.**
- **S4B7: Retrieve behaviour and article data from a database.**
- Establish the requirements for switching out to a potentially improved machine learning model.

S4F1: As a user, I want to see more recommended articles under a clicked article to keep me engaged.

As mentioned in the initial problem description 1.1, the main issue Ekstra Bladet is facing is keeping its users engaged, and this is one of the key features to mitigate this issue.

The PBI implements a component at the bottom of an article to help users find new articles. An initial illustration was presented to the product owner, Hartwig, for feedback.

Anbefalet til dig



Figure 5.4: Initial illustration of the feature for this PBI

The design was primarily influenced by similar swiping interaction components on news apps like The Guardian's [54] and the dating app Tinder [55], given its iconic and highly influential use of swipe cards. This PBI features a single element in focus with the next elements placed in the background, accessible by swiping upwards.

S4F2: Store and continuously update user behaviour tracking data in a database. Save both as the format provided by Ekstra Bladet, and the format used in the model.

When working with a collaborative filtering model, the relevance of the dataset is eminent. The dataset is getting expanded with new articles and user behaviour entries every minute as users browse, click and constantly create impressions. This means, that the model must be trained on the new data periodically so it can rank and predict more precise, up-to-date, and relevant news.

The user data is stored from the application in the local and cloud database every time a user enters and exits an article. The data must be saved in two different formats. The original format of what is given by Ekstra Bladet and the format which LightFM can use for training. Only the LightFM format is actually used in the training cycle and retraining the model. The Ekstra Bladet format continues to be saved even when not actively utilised, as it allows for future model changes without losing the valuable user behaviour data accumulated during the implementation of LightFM.

S4B1: Implement model serving.

The majority of model serving has already been implemented throughout previous sprints. This time, everything is checked to adhere to the good practices and common setup standards described in the technology stack section [4]

The missing parts, namely being able to switch models out, retraining models, and evaluating the model using the API have all received their respective PBIs.

In this PBI, the FastAPI file and its utilities are cleaned up and their purpose and functionalities are made clear.

Listing 5.6: LightFM API File

```
1 recommender_system = RecommenderSystem(data_path, models_folder_path)
2 recommender_system.load_data()
3 recommender_system.load_model(model_id);
```

When starting the LightFM API file, an instance of the recommender model is made, passing the data path for any locally stored files, and model folders path - more about that later. After that, the data and model are loaded and the recommender model is ready for use.

S4B2: Enable developers to switch between different models without rewriting the API code.

During model serving, it is important to streamline the process of switching the machine learning model out, because that is a common action.

As can be seen before, a recommender instance receives a model folder path when instantiated.

If the model is to be switched out, a new one just has to be loaded.

Listing 5.7: Switching models out

```
1 def load_model(self, model_id):
2     model_file = os.path.join(self.models_folder_path, f"{model_id}.joblib")
3     if not os.path.exists(model_file):
4         raise FileNotFoundError(f"Model file not found for model ID: {model_id}")
5
6     # Load the model
7     self.model = joblib.load(model_file)
8     print(f"Loaded model {model_id}")
```

When the `load_model` function receives the model name, it is joined with the given model folder path and appended with `".joblib"`. If the model cannot be found, an error is thrown before attempting to load.

In the future, for example, if a good interface is created, this load function could easily be hooked up to an API call just like other functions already are.

S4B3: Implement model retrain for a served model.

Implementing model retraining on new data is also an important functionality that should be supported in the model serving layer.

The implementation is a complete retrain, because of LightFM's limitations regarding partial fit. When using LightFM's partial fit function, the users/items/features in the supplied matrices must have been present during the initial training. This means that if new users/items/features are added to the dataset, a full retrain is required. Ideally, a partial fit would be the better option, not losing any previous model training progress.

Listing 5.8: Retraining the model on new data

```
1 def retrain(self, epochs):
2     self.load_data()
3
4     # model learning rate
5     LEARNING_RATE = 0.25
6     # no of latent factors
7     NO_COMPONENTS = 20
8     # seed for pseudonumber generations
9     SEED = 42
10
11     self.model = LightFM(loss='warp', no_components=NO_COMPONENTS,
12         learning_rate=LEARNING_RATE, random_state=np.random.RandomState(SEED))
13     self.model.fit(interactions=self.train_interactions, epochs=epochs);
14
15     joblib.dump(self.model, 'Saved_Model/lightfm_model_retrained.joblib')
```

In short, new data is fetched, a new model is initiated and trained, and the retrained model is exported. Currently, the only passed argument is the number of epochs, but all other configuration figures could easily be included in the request instead of being hard-coded as it is at the moment.

S4B4: Implement performance monitoring for a served model.

Evaluating the model uses the same AUC score LightFM module that was already implemented in the model training layer of the system, Just like in the model training layer, train interactions are excluded from testing to ensure correctness.

Listing 5.9: Evaluating the loaded model

```
1 def get_validation_AUC_score(self, num_threads=1):
2     test_interactions_excl_train = self.test_interactions - self.train_interactions
3     .multiply(self.test_interactions)
4
5     auc_scores = auc_score(
6         self.model, test_interactions=test_interactions_excl_train, num_threads=
7         num_threads,
8     )
9     average_auc = np.mean(auc_scores)
10    print(f"Average AUC Score: {average_auc}")
11
12    return average_auc
```

When running the code, the resulting AUC is outputted in the terminal. This is quite a crude solution, but would be very easy to append the score to a file or database entry.

S4B6: Adjust training parameters to improve model performance.

It was decided to experiment with adjusting the parameters with the primary goal of improving the immediate performance of the recommendation system. Thinking longer-term, one should also take generalisability into account, meaning the model is not overtrained on- and only performs well with the data the development team currently works with. Also, monitoring long-term performance fluctuations would be important considering long-term performance.

This was done by preserving a chosen model which was ensured to run correctly, and then creating a copy of that model where the chosen parameter changes were made. The changes in the AUC score were documented and weighed against

each other to decide which change should be implemented permanently going forward.

The default model which was chosen as the standard of these tests had the following values assigned to the parameters.

Listing 5.10: Parameter values assigned in the default model

```
1 # default number of recommendations
2 K = 10
3 # percentage of data used for testing
4 TEST_PERCENTAGE = 0.25
5 # model learning rate
6 LEARNING_RATE = 0.25
7 # no of latent factors
8 NO_COMPONENTS = 20
9 # no of epochs to fit model
10 NO_EPOCHS = 20
11 # no of threads to fit model
12 NO_THREADS = 32
13 # regularisation for both user and item features
14 ITEM_ALPHA = 1e-6
15 USER_ALPHA = 1e-6
16 # seed for pseudonumber generations
17 SEED = 42
```

The result of these values was the following AUC scores per epoch.

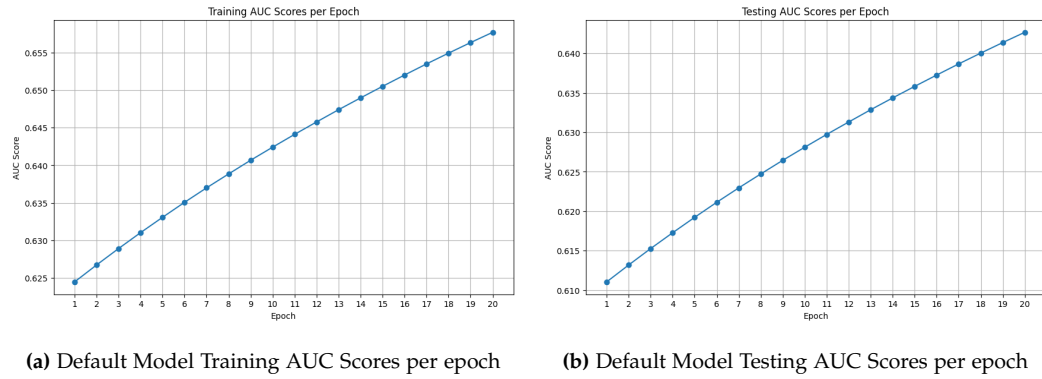
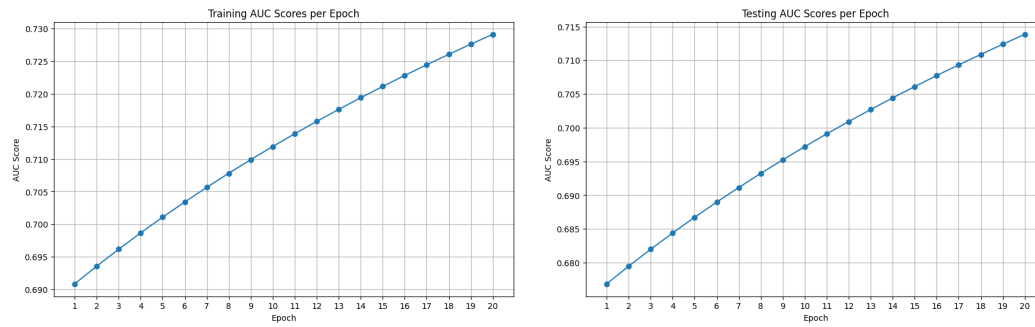


Figure 5.5: AUC Scores per epoch for Training and Testing

The learning rate determines the step size at which the model parameters are updated during training. A higher learning rate can lead to faster convergence, but it may also cause the model to overshoot the optimal solution. [22] This was investigated by reducing the learning rate from 0.25 to 0.1, which resulted in a 31% increase in the AUC score. This dramatic increase in performance seems unusual, as changes in hyperparameters should lead to smaller incremental improvements first noticeable when nearing convergence rather than such a massive leap. Therefore, this change would not be considered going forward, and the default value is kept.

The latent factors represent the underlying features of users and items in the model. [60] By adjusting this hyperparameter, it could allow the model to capture more complex patterns in the data. By increasing the number of latent factors from 20 to 30, an AUC score reduction of 2% was found. This worse performance could be due to overfitting, where the model becomes too complex and starts capturing noise instead of meaningful patterns. As this change resulted in the model performing worse, it was decided to experiment with the opposite change to investigate a potential increase in model performance. By reducing the number of latent factors from 20 to 10, the following AUC scores were found.



(a) Training AUC Scores per epoch with decreased number of latent factors (b) Testing AUC Scores per epoch with decreased number of latent factors

Figure 5.6: Comparison of Training and Testing AUC Scores per epoch with decreased number of latent factors

This change in performance is not drastic enough to be unusual, as was the case with the adjustment in learning rate. However, this change is not subtle enough to the point that it should be ignored. Therefore, a change in this hyperparameter will be considered as a permanent addition going forward.

To ensure the model is still behaving as intended with the change of this hyperparameter, there was a need to find convergence. This is the point where the model's predictions stop improving, or the error rate becomes constant. If no convergence exists, then there is no reachable point of stability for the model. To reach convergence, the number of times the algorithm would iterate over the training data to learn the underlying patterns was increased to 1000 epochs. The result of this showed no convergence, but the point of convergence seemed reachable. However, to increase the amount of epochs, an improved structure of training the model would be optimal. These optimisations will be implemented and addressed in the upcoming sprint.

S4B7: Retrieve behaviour and article data from a database.

Up to now, the model used a locally stored version of the EBNeRD dataset. This static setup was not fit for continuous retrains. Therefore, the connection to the database from the model serving instance has been created. It fetches the data in a certain range from the LightFM database table and replaces the local dataset. When LightFM outputs the article IDs as predictions, a query to the database is made to get articles from the articles table, which can be sent to the frontend.

5.7.2 Sprint Review

The system is now complete, featuring a model that performs relatively well thanks to configuration changes, achieving an average AUC of 0.713 on the validation dataset. Still, this score can be improved with more training. Additionally, there have been meaningful enhancements to the model serving layer, including completing a continuous UX flow on the frontend with additional recommendations under clicked articles, and finalising operations related to the database layer.

Ensuring the system's success in this project relies heavily on its modularity, necessitating thorough documentation of any meaningful constraints. A particularly important aspect is how simple it is to grab an improved model and put it in place of another. With the current setup, any joblib formatted model can be loaded using an argument if they have an `auc_score`, `fit`, and `predict` functions with correct arguments. The function `auc_score` should receive the model and test data, `fit` should receive the train interactions and the number of epochs, and `predict` should receive a user ID and item IDs for those items that should be included in the predictions.

Overall, this is not a big ask, as these elements are present in virtually every

recommender system. The joblib file format requirement does not seem to propose any issues either, other than having to write some additional code so one's model is not only exported as a TensorFlow or PyTorch model but also as a joblib file. Although it is already common practice, so the technology should not be completely foreign to developers.

Shortly before the end of the sprint, Ekstra Bladet's data science team lead, Jesper Rix assisted the development team by describing Ekstra Bladet's MLOps setup in an email [9](#). The following sections are a summary of the contents of the message.

Currently, Ekstra Bladet is using a FastAPI/Flask model serving setup largely identical to this project's. It is worth noting that Ekstra Bladet is planning a transition to a Kubernetes/Argo Workflow setup using PyTorch Serve or Nvidia Triton Inference Server. Kubernetes was also previously researched and described as a fancier model serving framework for larger organisations. CI/CD-related model operations, Docker container hosting, and cloud services are all done using AWS solutions. Otherwise, tests and deployment are automated using GitHub actions run upon merging.

Ekstra Bladet's collaborative filtering models are trained every 2 hours as they otherwise have "cold start" problems as the extra magazine publishes many articles in 2 hours. The content-based models are trained once a day and are more robust and can handle cold starts. Tests have been run where the models were only trained once a week vs. every day, and there was only slightly better performance by training every day. The data science team has not looked at metric-based retraining of the models (for example retraining after X number of new articles published).

The only metrics used for production-level model performance evaluation are

sales and click-through rate. Nevertheless, Ekstra Bladet collects a lot of data about the models' performance and their derived effects in the news feed, but it's not something they currently use for monitoring, only plan to.

With only a single sprint left, the development team will not attempt to recreate the MLOps setup Ekstra Bladet wishes to transition towards. Still, the provided information is highly valuable in confirming that the development team's solution is, arguably in many aspects, on par with the currently deployed system.

Having delivered a complete system, it will now be important to clean up the system in terms of code quality and modularity. If needed, additional tests should be written. Furthermore, the main takeaways regarding the model training layer should be streamlined and presented more systematically.

5.8 Sprint 5

The main focus of this sprint is to finalise the product. This means that no major additions to the product are planned, only cleanup work and solidifying the stability and code quality of the product.

5.8.1 Sprint Planning

The following is a list of the backlog items which are addressed during this sprint.

Front-end items

- S5F1: As a user, I want to see more recommended articles under a clicked article to keep me engaged. (Improving Stability)
- S5F2: Write any missing unit tests to cover more complex components.

- S5F3: Perform a general code cleanup.
- S5F4: Clean up the folder structure and file names.
- S5F5 Perform static code analysis using CodeScene and fix any potential issues.

Back-end items

- Identical to frontend items F2-F5

S5F1: As a user, I want to see more recommended articles under a clicked article to keep me engaged. (Improving Stability)

After gathering additional feedback from the meeting with the product owner **5.7.1** the development of the PBI started in sprint 4 and concluded in sprint 5.



(a) The carouselCards component at the bottom of an article



(b) The tutorial overlay displayed on top of the first card in the stack

The user is presented with articles represented in a stack of cards, which can be navigated through by swiping upwards or downwards. When a card is tapped, the user is navigated to the selected article. Additionally, an assistive overlay was also created as the result of a short test with a supervisor. The overlay is placed on the first card and instructs the user to swipe up then fades out when a swipe-up gesture is detected.

In terms of optimisation and improvement regarding this backlog item, some future work could be considered. The articles displayed in the wheel are, as of this sprint, passed on from the home page and display the same articles. However, the component was designed to be modular, making it easy to change the data which is being displayed in the stack of cards. In the future, when a function is created to gather recommended articles based on the currently viewed article, the articles can simply be passed onto the component by using the data input parameter.

5.8.2 Sprint Review

The primary objectives of this sprint were successfully achieved, focusing on enhancing stability and code quality. Through thorough clean-up efforts and improvements in modularity and maintainability, the codebase underwent significant refinement. Additionally, minor adjustments to other tasks were finalised, contributing to a solid foundation for future enhancements.

Chapter 6

Quality Assurance

To ensure and evaluate the quality end product, the development team reflects on the system setup in light of theoretical correctness, performs static code analysis of the codebase using CodeScene, and uses the AUC score to provide a conclusive evaluation of model performance. Furthermore, other quality-assurance-related aspects of workflow are discussed and how they could be extended to ensure higher product quality.

Quality Assurance, QA, is the event of ensuring various processes, procedures, and standards are correct and suitable for a given project. [18] The implementation of QA-related processes is parallel to the development of the project, thus resulting in faster development time with fewer recurring bugs. Known bugs are documented as GitHub issues, and are most commonly addressed with hotfixes.

Throughout the project, sprint reviews served as a quick quality check on the completed product backlog items and delivered increments, with the product owner having the last say in accepting items to be considered done or not. Their input was imperative in producing meaningful increments with acceptable quality. No user tests were conducted due to the massive effort it would take to pro-

duce representative data from the user feedback. Instead, knowing the product owner’s professional experience in their field, they were responsible for representing their users’ interests.

6.1 Evaluation of the system

The product, as stated in the problem statement 3.4, is a modular system with a mobile frontend which supports a machine-learning-based recommender model and database communication. Software modularity is a concept, which uses sets and elements to explain the different modules of a software architecture. [25] The way to measure the degree of modularity is by reviewing the internal module cohesion and the degree of interdependence between modules. For simplicity, the layers from the system architecture, 4.6, are treated as the modules.

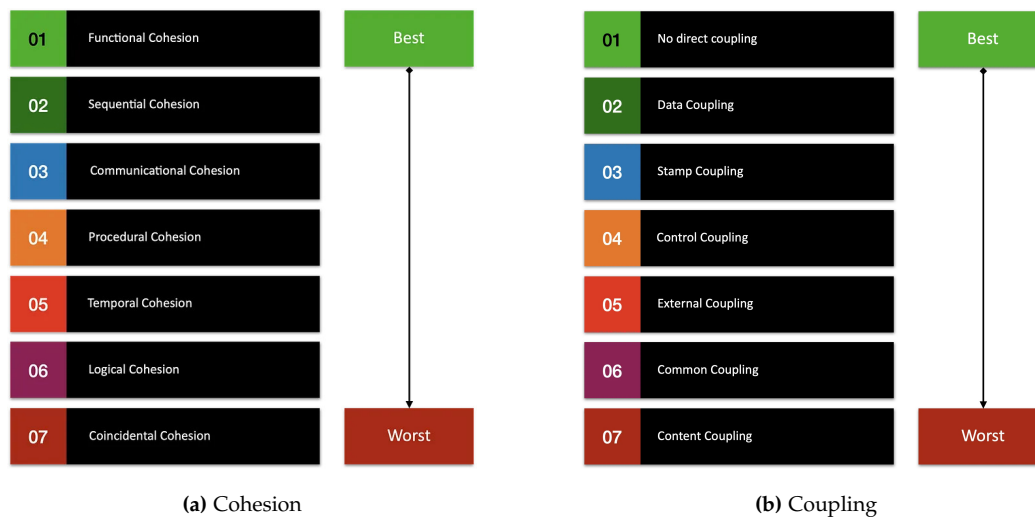


Figure 6.1: The different levels and types of cohesion and coupling

6.1.1 Module cohesion

Module cohesion can be defined as the following: *"In computer programming, cohesion refers to the degree to which the elements inside a module belong together"* [25], and there are different levels and types of cohesion, which can be seen in figure

6.1a When measuring cohesion, it is vital to determine how the elements inside of the module are grouped and what their relationships are to each other.

This project's implementation of high cohesion is achieved by separating the code into layers from the system architecture. Inside the application layer, the functionality is divided into sub-modules, which handle local storage, database, API and user interface-related components. The data layer has only database-related components, whereas the model serving only contains model-related functional components. The modules have functional cohesion since all the layers in the system are designed to handle their domain of tasks. Thus, it is easier to maintain their interchangeability in the future.

6.1.2 Module coupling

Coupling can be defined as the following: *"In software engineering, coupling is the degree of interdependence between software modules; a measure of how closely connected two routines or modules are"* [25], and as cohesion, there are multiple types of coupling as seen in figure **6.1b**. Low coupling is an indicator of a well-structured system, and it is the goal when designing a modular system.

In this project, there is data coupling which is the next highest type of coupling. The modules share data through parameters or API calls, where only the necessary data is passed through. This means only when there is a complete restructuring of the data, there is needed to alter the function to handle the change. In that case, it would be good practice to add a data model layer. This ensures that the data always has the same format when being sent to/from the application layer. An example of not having a data model layer is several functions in the database sub-module in the application layer that require the data created from the local storage sub-module as input to send to the database. Ideally, all interactions between sub-modules would go through one channel to minimise the

changes in the modules if necessary. This would result in less interdependence between the modules and then a lower coupling.

6.2 Code quality

The communication between the different components in the system is done by using an application programming interface, API. Thus, the API and the components must be tested separately and together to evaluate and ensure the reliability of the system. The system has so far been separated by both frontend and backend and system sub-layers. When focusing on the code quality and testing, the system is therefore separated into frontend and backend.

In the frontend, the following components are tested; Axios requests, asynchronous storage (React Native local storage), and database operations. These components were prioritised for testing because they are responsible for the most meaningful communication and storage functionalities. Axios requests are tested since it handles sending and requesting articles from the model. If this component is failing for some reason, the whole app is useless since there are no articles to read. The asynchronous storage and database components are closely intertwined because all database functions are called from async storage functions. Async storage is tested to ensure that all the user data is correctly saved locally, whereas database functions test that sending the user data to the cloud database is successful. Considering that the model object in the backend fetches the newest user data from the database, the database functions must be guaranteed to work as expected.

Considering the backend, the most important components to test were the API endpoints and model-related utilities.

To test the endpoints, mock payloads are sent to the API, and assertions are made for the format of the returned data and status code.

To test the model-related utilities, a fixture is initiated, creating an instance of the model. Thereafter, the functions responsible for loading data, loading a model, evaluating a model (AUC), retaining the model, and making predictions using a loaded model are independently tested with mock data and individual unit tests. Furthermore, to ensure correctness in real-life scenarios, a series of these core functionalities are tested for example containing a sequence of the same operations that would be run when switching a model out and making predictions, or when performing a retrain call. With these more complex tests, the correctness of multiple functions working together is also checked.

The data preparation file, which converts Extra Bladet's raw behaviour and history into usable data for the LightFM model, is tested with numerous sanity checks throughout the process. Despite this file only having to be run once, the generated data must be correct and usable for proper model training. Learning from Ekstra Bladet's issue of having overlapping items present in both the training and validation datasets, the development team took a lot of precautions when preparing the data.

If the testing of data preparation received more attention, it would have probably received a package of tests that could be run on the exported data. Knowing that LightFM is strict about its data, for example evaluating with overlapping data is not allowed and the internal mapping mechanism throws a warning when conflicting click/no-click entries are received, it can be assumed that several checks are also run on the data before model training begins.

A CodeScene static code analysis was run on both front and backend reposi-

tories to establish general code quality. This analysis should highlight issues related to code complexity, tight coupling, dead code, and other more intricate things contributing to technical debt. [44] The results strongly suggest that the code is readable and high-complexity problems are successfully broken down into more manageable sub-parts. The frontend repository has scored an average of 9.84, and the backend repository has an average of 9.78 out of 10. No changes are explicitly suggested by the static code analysis tool, the most "problematic" file being a file for testing endpoints in the backend, where the tool indicates that there may be a few too many functions in a single file. This is looked at by the developers, but no changes are made because keeping all of these tests in one place seems like a more solid option than spreading them out into different files.



Figure 6.2: Summarised results of the CodeScene static code analysis

6.3 Model Quality

Both in the model training and model serving layers, the AUC score was used as the primary metric for model evaluation, as it is the industry standard for machine-learning-powered binary classification recommender systems. [2] For calculating this score, LightFM's own ROC AUC score module was used. As previously described [2] LightFM has numerous automatic checks for potential data leaks, and the development team has done numerous sanity checks to ensure that the EBNeRD dataset converted to a LightFM-compatible format has no issues.

The most optimal LightFM configuration found by the development team uses a 75-25% test split with a learning rate of 0.25 and 5 latent factors. Training with this setup for 4000 epochs yielded a training AUC score of 0.9082, and a testing AUC score of 0.9001. This means that 90% of the predictions made by the model should be relevant to a given user. By observing the plot, it can be seen that the model reaches convergence, although does not overfit yet. Given that every 100 epochs only yielded a constantly falling increase of 0.0001 in testing AUC score, so the development team did not spend more time on the model, but instead explored more configuration options. After running an identical configuration with 10 14 and 20 15 latent factors for 4000 epochs each, it was also determined that given enough time, these configurations would also top out at around the same AUC scores as the one with 5 latent factors.

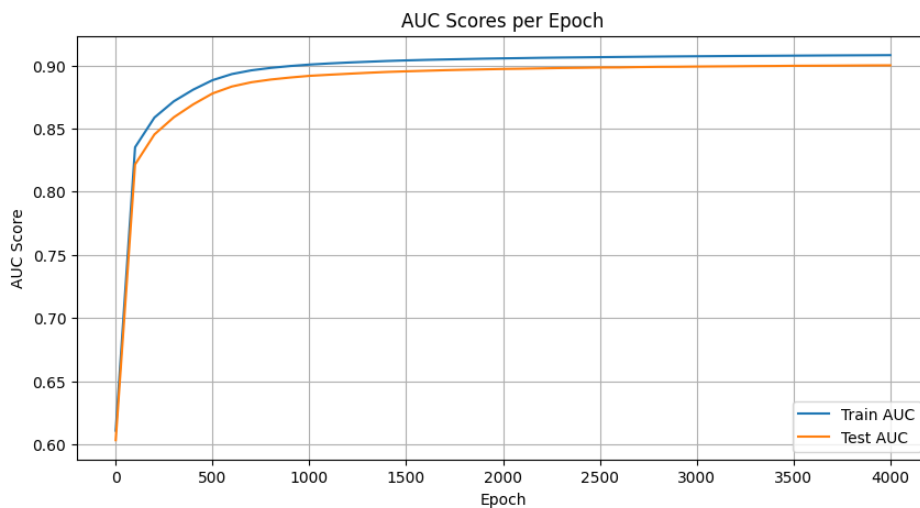


Figure 6.3: Results from the most optimal LightFM model configuration

LightFM provides a module for cross-validation when generating training and test splits, and the development team ensured that different random seeds do not have any meaningful effect on evaluation results ($\pm 1\%$).

Initially, the F1 score was also used as an evaluation metric, but it was not im-

plemented when switching to LightFM, because it was not considered a high priority. If the development team had a bit more time, LightFM's "Precision/Recall at K" evaluation modules could have also been implemented fairly easily. By looking at the documentation[50], these two modules look like utilities for a model-specific implementation of NDCG. 2

Lastly, another intuitive way to determine the model quality is to compare this project's model to other models on the same dataset. The EBNeRD dataset is obtained from a machine-learning challenge held by Ekstra Bladet called RecSys Challenge 2024 [12]. Both this project and the RecSys challenge primarily measure a model's performance by AUC - NDCG could have been used as a secondary metric, as it is also used in the challenge. As previously described, implementing NDGC should be possible with LightFM utility modules.

Rank	Team	AUC	MRR	NDCG@5	NDCG@10
1	taksai	0.8336	0.6409	0.7072	0.7257
2	taksai	0.8306	0.6414	0.7057	0.7244
3	taksai	0.8279	0.6371	0.7013	0.7203
4	taksai	0.8226	0.6285	0.6929	0.7135
5	carlosgaravatti	0.8221	0.6195	0.6879	0.7087
6	asato	0.8211	0.6262	0.6907	0.7118
7	doubleq	0.8201	0.6178	0.6870	0.7076
8	tomoyukiarai	0.8196	0.6237	0.6884	0.7098
9	axi2017	0.8193	0.6188	0.6847	0.7076
10	axi2017	0.8183	0.6166	0.6826	0.7057

Figure 6.4: RecSys Challenge 2024 Leaderboard [28]

As seen in figure 6.4, the highest model submission has an approximate testing AUC score of 0.83. The challenge most likely uses a different validation dataset

(to ensure that a participant's model is not trained on it), so when comparing the leaderboard results to this project's model, it can be assumed that the LightFM configuration would produce similar results.

6.4 User validation

Performing user testing could have arguably proposed numerous potential improvements on the frontend side of the application, or have given more ideas for different ways of implementing a personalised feed. After all, user satisfaction and engagement are core qualities which should be taken into account when deploying such a massive overhaul of a mobile application.

Given that the development team was given an exact design from the start, the product owner already had a clear vision in mind, which seemed more than reasonable for the developers.

Although evaluating whether a personalised news feed results in increased engagement compared to a general news feed would mean a lot considering the success of the project, doing so would require a massive number of test participants to produce representative results. Most realistically, it would be tested in production.

Certainly, Ekstra Bladet has the resources to conduct enough testing to be able to produce representative data, but due to the lack of time or meaningful/representative/quantifiable results, the development team chose to not put resources into a small-scale user validation session.

Chapter 7

End product

This chapter presents the final product, showcasing the culmination of our project's development. Key features are highlighted, and a high-level technical overview is provided.

The end product is a system consisting of an overhauled mobile application (React Native with Expo), where Ekstra Bladet's articles are presented to users in a personalised news feed. These articles are recommended by a well-performing collaborative filtering model (LightFM model with 0.9 AUC score) trained on Ekstra Bladet's interaction data. While using the app, user interactions are stored in- and article information is retrieved from a cloud database (Supabase).

7.1 Mobile Application

7.1.1 Personalised News Feed

When the application is opened, the users are presented with a personalised news feed. The contents of this feed are generated by sending a request to an API that serves a machine-learning model to make recommendations and fetching the recommended article data from a Supabase database. The current model is an

implementation of LightFM trained on interaction data from the EBNeRD dataset provided by Ekstra Bladet.

7.1.2 Article Interaction

When the user has found an interesting article, they can tap on it to read it. The application navigates to a separate screen which displays the contents of the article in a readable format. Additionally, a progress bar indicates how far the user has read in an article and, when on the feed, displays the reading progress for previously read articles.

7.1.3 Swipe-able Article Cards

When a reader reaches the bottom of an article, they are presented with a stack of cards, where each card displays a preview of another article, including the title and thumbnail image. Each card displays an article recommended to the user, and if the currently displayed card is not found interesting by the user, they can discard it by swiping up, revealing the next card underneath.

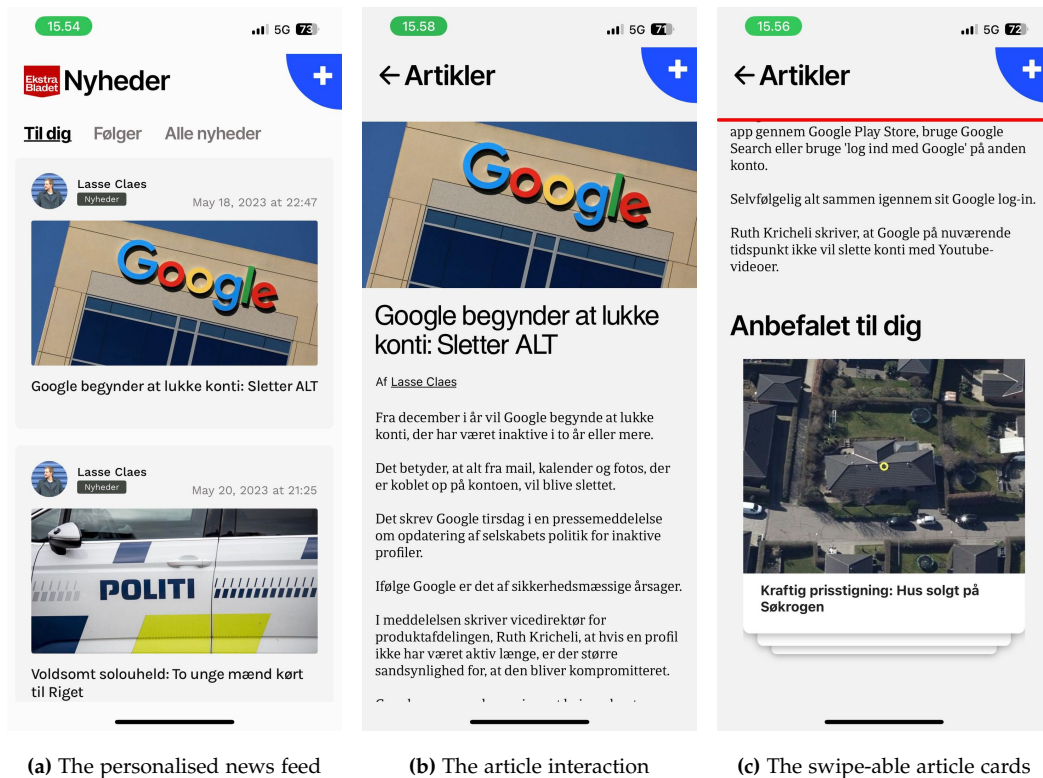


Figure 7.1: Key features of the mobile application

7.1.4 User Behaviour Tracking

The application continuously tracks article clicks (or no-clicks), read times, and reading percentages. The tracked interactions are stored both locally and in a cloud database corresponding to an automatically generated userID. This is important, because to continuously recommend relevant articles, user behaviour must be tracked and the model must be retrained on the new data. To align with Ekstra Bladet's standards, and also to skip a computationally intensive data-preparation step for the currently implemented LightFM model, interaction data is both saved in Ekstra Bladet's format and a format compatible with the LightFM model.

7.2 Model Serving

As mentioned before, recommendations are requested by the application through an API call to the model-serving API. Model serving is done using the FastAPI RESTful API framework. By making requests to the specified API endpoints, the application can retrieve a list of the latest news or a specific number of recommended articles, and developers can easily switch models out as long as they are serialised as joblib files, and have an `auc_score`, `fit`, and `predict` functions with correct arguments. Developers can also evaluate (get AUC) and retrain models for a requested amount of epochs by reaching the correct endpoints.

7.3 Model Training

Before a model is served (used in the application), it must first be trained and evaluated. The training process for the LightFM model is handled in three major events. First, the data is preprocessed, meaning the user behaviour and article data are formatted to be compatible with the model. Second, the model is trained by running a specified number of training epochs and evaluating the model performance using AUC scores for training and testing (these scores are also plotted to give a better overview). Lastly, the model is exported as a package including the model, a configuration file containing all of its training variables (e.g. learning rate, number of epochs, latent factors, etc.), and a CSV file containing evaluation data (training and evaluation AUC scores and corresponding epoch number).

7.4 Cloud Database

When the application requests recommended articles, the model serving API returns a list of article IDs, and the article data is fetched from a Supabase database.

When initiating a model retrain using the model serving API, new user interactions are fetched from the Supabase database. As previously mentioned, user interaction data is also stored in two formats. The first format is the EB format and the other one is the format of what the current model accepts. Both are continuously updated using API calls.

Chapter 8

Discussion

After finishing up product development, there is naturally always room for improvement and countless ideas for how the product could be scaled and its quality refined. Other than exploring possibilities for future enhancements, the success of the project and overall workflow are discussed, and bigger-picture ethical concerns and potential risks of the product are explored.

8.1 Reflections on the development process

This section encapsulates a critical analysis of the development team's Agile workflow, events, cooperation between roles, and pipeline.

Overall, the use of 2-week sprints consisting of sprint planning, semi-daily internal stand-up meetings, and sprint review was very helpful in encouraging the development to continuously deliver valuable increments and to keep the development on track whenever the focus shifted from the main objectives of the project. The lack of formal sprint retrospectives did not give the impression of insufficient continuous improvement considering the process itself, given that the development's workflow and morale were largely unchanged, and never resulted in a failed sprint. Arguing from a theoretical perspective, formal sprint

retrospectives may have resulted in increased value delivery, but at no point felt like a necessity.

Sprint planning aided a lot in getting an overview of the coherence between the main goal of a sprint and how its smaller parts (PBIs). Based on experience, the development tackled documented dependencies during parallel development by using mock data, bypassing unnecessary navigation in the app, and working with older machine-learning models. The solution's degree of modularity also enabled the development to switch components out with relative ease.

Daily stand-up meetings helped keep every developer in the loop with others' progress, and at times resulted in useful updates and questions for both supervisor and product owner. During sprints, updates were frequently sent out to ensure transparency between parties.

Physical sprint review meetings were held separately for the product owner and project supervisor. Each event entailed a presentation of the sprint's PBIs, any relevant new findings, gathering feedback, and asking/answering questions from both sides. After each sprint review, the development left with a clear goal in mind for the next sprint, and all notes taken throughout the event were immediately formalised in a to-do list. It is worth mentioning that the product owner was more invested in the application and overall system than specific model developments, so these aspects received more attention in presentations tailored for them.

Since the development team is writing this report, it is difficult to give an unbiased evaluation of how well the development fulfilled their role. Considering the most important aspect of Agile - delivering value with an acceptable velocity, the development did succeed in fulfilling their role, while ensuring good communi-

cation with other actors and continuous integration of the stakeholders' feedback.

The supervisor of this project did an outstanding job at mentoring a development relatively new to machine learning, and initially entirely clueless about recommender systems. Other than consistently providing helpful guidance and clear feedback throughout the entire project period, they ensured that the development was always working towards answering the central problem statement of the project. Communication was never an issue, and the supervisor willingly took extra time to answer questions outside of meetings, further highlighting their unwavering support.

The product owner of this project provided the group with a clear product vision from the start. Given this, the majority of the application itself was already finished in approximately 2 sprints, and they were more than open to discussing new ideas. During meetings and discussions about the product, their feedback was concise and user-centric, providing a very useful perspective for the developers. The product owner being more invested in the application and seeing the overall system in action would at times mean that the acceptance of some model-related developments partially relied on the supervisor's feedback. This should not be interpreted as a shortcoming on the product owner's side, because the product owner should be a functional expert, not a technical expert. Communication was never an issue, and the product owner was always actively involved in the process, often contacting technical experts from Ekstra Bladet's data science team to provide even more support for the development team.

The workflow aspect that should have received more attention was the continuous integration through GitHub actions. The only automated check is linting, and not using GitHub actions for running all unit tests, building the app, and for example, performing static code analysis on a push, are all missed opportunities

that could ensure better code quality and not having to spend time on minor hot-fixes later. Still, there were only very few instances where bugs slipped through the code review process.

8.1.1 The product owner's reflections on the development process

As mentioned in the previous section, there could be a risk of a biased evaluation if only the development team's perspective was documented. To alleviate this, the product owner was contacted with questions regarding their thoughts on the cooperation throughout the project, and their sense of fulfilment of fulfilment as a central actor. [9] Regarding communication, they stated that they felt there was a level of transparency which exceeded their expectations based on their experience with previous projects within the same problem domain. Regarding their sense of fulfilment as a product owner, Hartwig stated that they felt a clear response to their feedback at all times and were satisfied with their influence over the end product. When reflecting upon the current product management methodology used at Ekstra Bladet, which focuses on leniency in managerial oversight, [41] the product owner felt they might have had too much of an impact on the product.

8.2 Technical difficulties

This section encompasses a discussion of issues strictly relating to the technical side of the project, mainly focusing on how the coding process was affected.

LightFM, being a best-effort service, has some limitations and issues not explicitly stated in its documentation. The fact that partial fit (used for incremental retraining) does not allow any new items or users to be introduced compared to the original training dataset renders it pretty much useless in production - although still very useful in the model training layer. Finding out about this other similar limitation through GitHub issues and not the official documentation

was at times frustrating. Furthermore, as formerly described, the recommenders package uses some tools readily available on Linux, but not on Windows or IOS systems, which forced the development to run the model training on Google Colab instead of doing it locally. LightFM does not have a GPU implementation, and disconnects were generally not an issue, so this did not have a major effect on productivity.

Other than LightFM, no major package-related issues were discovered, and Expo was a helpful tool for keeping all dependencies up-to-date. The only minor issue requiring a hotfix was when updating Expo SDK from 50 to 51, the react-navigation package required an additional import in the app.js file to work properly. Luckily, this was a documented issue, so the solution was straightforward to implement.

8.3 Future work and scaling

While the end product arguably provides an adequate solution to the proposed problem, both the development team and product owner had numerous ideas about how the system could be scaled with additional features and enhancements to existing ones. Other than theory-crafting new ideas, there are also some leftover product backlog items which have not been addressed due to receiving lower priority or requiring too much time to complete. In this section, the possible future work on the product is detailed.

8.3.1 The product owner's thoughts on future work

The product owner was contacted to gather their thoughts regarding future work, which is crucial, as it assists in aligning business goals, prioritising the tasks and setting clear expectations for all parties. When asked about what aspects of the product should receive the most focus if the development team was given more

time, the product owner responded "The news experience". ⁹

Hartwig describes the potential of implementing machine learning as something which can curate an interesting news feed, instead of relying on a "news reaction", which is gathered by "clickbait, sensations and doom-scrolling". The product owner thinks it is a shame that their current focus is on creating clicks, rather than having the user provide them naturally through curated content. However, they are unsure if the source of this issue stems from the news feed, or the articles themselves.

The product owner has a clear vision for the issue which needs to be tackled, and they also describe candidates for the root of these issues. Regarding which should be handled first could become clearer by discussing resource management in the next iteration, while also discussing possible solutions to these issues as they require different tools to solve.

8.3.2 Leftover PBIs

As described above, some PBIs were planned, drafted and loosely defined but did not get implemented due to time constraints or a shift in priorities. PBIs with significance will be expanded upon beyond the listing.

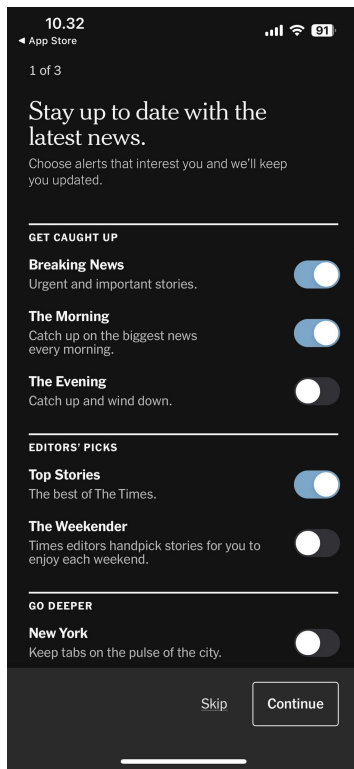
- Update the carousel cards component to use content-based filtering when fetching data
- Once a model is Content-Based Filtering hybrid is implemented: Make frontend for onboarding (cold start solution)
- As a user, I want to see different types of articles (live, breaking, gallery, etc.) with unique styling
- As a user, I want to be able to bookmark news

- As a user, I want to have a history of my viewed articles and how much of them I have read
- Make an example of an ad in the news feed
- As a user, if there is no image provided for an article, I want to see a large text thumbnail
- As a user, I want to be able to follow events (use named entity recognition data)
- As a user, I want to be able to switch between light and dark display options

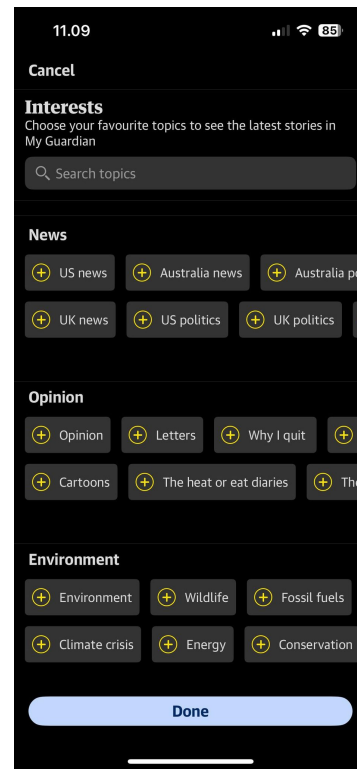
Once a model is Content-Based Filtering hybrid is implemented: Make frontend for onboarding (cold start solution)

This PBI would be picked up once a Content-Based Filtering model has been implemented. The purpose of this PBI is to solve the aforementioned cold start problem, which involves a new user whose initial preferences are not specified and is therefore difficult for them to receive recommendations. [2.3.1](#)

As for the design of the onboarding solution, other news outlets use similar designs, with various topics displayed to the user to which they respond by pressing the ones they want to see more often.



(a) The New York Times app



(b) The Guardian app

Figure 8.1: Onboarding on two different news apps

In the case of Ekstra Bladet, their article data contains a property which specifies the article's topic. This same property could be used to display all possible topics to the user, which they can then pick the ones they want to follow.

Update the carousel cards component to use content-based filtering when fetching data

As mentioned in the section regarding the component, the carousel cards component, which is supposed to recommend articles based on the currently viewed article, currently uses the same collaborative filtering model as the personalised feed. In this context, a content-based filtering model would make a lot more sense to implement.

8.4 Ethical concerns

The success of Twitter (X), Instagram, and most recently TikTok has proven that a "for-you" page providing relevant data for users can result in a massive boost in engagement longevity and retention. [1] While this unarguably has a good effect on business value, it can have dangerous side effects.

When browsing non-personalised content, the user is met with content that does not necessarily align with their personal beliefs, political agenda, or religion. Users are involuntarily confronted with other perspectives, which may at times mean that a user has the opportunity to form a more well-rounded opinion on any given subject. On the contrary, when browsing content on a "for-you"-page, if an algorithm has gathered enough data to accurately map a user's personal preferences, (which are likely labelled by, for instance, political leaning and similar properties), they enter a so-called filter bubble.

A user being in a filter bubble implies that algorithms have segregated the user from information and viewpoints that diverge from their expressed interests, potentially causing them to overlook crucial information. This phenomenon can narrow the user's worldview and impede exposure to diverse perspectives, hindering their ability to make well-informed decisions and engage in meaningful discourse. [20]

This phenomenon is empirically documented in Leysen et al.'s research paper [5], concluding that *"(...) filter bubble-existed recommendation systems often decrease the diversity of users' beliefs, leading to bias and ideological segregation, reinforcing users' existing beliefs and limiting their exposure to alternative viewpoints (...)".*

As documented in Vozab's research paper [59], average media literacy is con-

tinually improving with younger generations, meaning that the older a user is, the more likely they are to be oblivious as to how filter bubbles affect their daily content consumption. Although this is a promising trend, media literacy in all generations is crucial for productive discourse.

The controversial use of personal data in politics is not unprecedented. Shortly after the 2016 United States presidential election concluded, both the Liberal and Republican parties were scrutinised for their extensive use of personal data to sway voters. The two parties spent a total of 72 million dollars on targeted Facebook advertisements, with Donald Trump succeeding most in increasing support among their potential voters, according to Ángel and Rubén Cuevas, researchers from the UC3M Telematics Engineering department. [29]

It must be stated, that the root of the controversy was how the data was collected - concerns regarding how data was used were the aftermath of the scandal. Namely, the root of the controversy is the Facebook–Cambridge Analytica data scandal, revealing that the personal data of millions of potential voters was collected and sold without their consent, and predominantly used for political advertising. [39]

Even if data is collected with the users' consent, who are very likely just clicking on mandatory GDPR consent-related popups without reading them, there are countless issues with targeted advertisements - or curated content - with effects on socio-critical events.

The closest any major platform has come to ensuring the factual correctness of advertisements is Twitter's community notes, where human moderators can take it upon themselves to reveal inaccurate information and missing context in any post including advertisements. These notes are displayed under posts, and those

who had previously interacted with posts that got a community note later on receive a notification too. [42] More recently, TikTok has also begun putting explicit labels on AI-generating content, as generative AI in combination with personalised feed introduces new complexities.

Chapter 9

Conclusion

The goal of this project was to provide a solution to the following problem statement:

Enhance the user experience and improve user engagement on Ekstra Bladet's mobile application by developing a highly modular software system that supports a machine-learning-based recommender model. Provide a complete system and display a personalised news feed to users on an overhauled mobile application. Prove the success of the machine learning with an adequate performance evaluation process using relevant metrics.

The end product of the project system encapsulates the core objectives outlined in the problem statement. The system leverages a personalised news feed for users as a result of the successful integration of a system that supports a machine-learning-powered recommender model. This is made possible by tracking user interactions in the application, storing them in a database, using this data to train a machine learning model, and deploying a model.

The recommender system, built using the LightFM model, has demonstrated

robust performance with a testing AUC score of 0.9, indicating its effectiveness in predicting user preferences accurately. Thus, although not supported by quantitative research due to time constraints, based on third-party research it is safe to assume that such a well-performing recommender greatly aids in enhancing user engagement.

The modularity of the system ensures scalability and future-proofing, allowing for easy updates and integration of new features or models. The overhauled mobile application provides an intuitive user interface and features like a virtually endlessly scroll-able feed and the swipe-able article cards further contribute to a continuous content flow.

The effectiveness of the machine learning component is validated through a comprehensive performance evaluation process. The model training and evaluation steps, including preprocessing, training epochs, and AUC score analysis, confirm that the recommender system meets the desired performance criteria. Additionally, the flexibility of the model-serving API, built with FastAPI, allows for easy deployment and management of the machine learning models.

While the project has met its initial objectives, there are several avenues for future enhancements. Potential improvements include refining the recommendation algorithms, enhancing the user interface, and integrating more sophisticated machine-learning models. Additionally, addressing leftover product backlog items will further enhance the system's functionality and user experience.

The ethical implications of personalised content and filter bubbles were considered. Ensuring that users are exposed to diverse viewpoints and preventing ideological segregation is crucial. Future iterations should focus on incorporating mechanisms to balance personalised recommendations with diverse content

exposure.

In conclusion, the project has effectively addressed the problem statement by delivering a complete and modular software system that enhances user engagement through personalised news recommendations. The combination of a high-performing recommender model, an intuitive mobile application, and a scalable architecture confirms the project's effort to meet its objectives. Moving forward, continued development and ethical considerations will ensure the system remains effective and responsible in its user engagement strategies.

Bibliography

- [1] *Adressa Dataset | Papers With Code*. URL: <https://paperswithcode.com/dataset/adressa> (accessed: 22.05.2024).
- [2] Anjali et al. *Layered Architecture*. 2024. URL: <https://www.geeksforgeeks.org/layered-architecture-in-computer-networks/>. (accessed: 28.05.2024).
- [3] Chuchan et al. *Neural News Recommendation with Attentive Multi-View Learning*. 2020. URL: https://www.researchgate.net/publication/334457560_Neural_News_Recommendation_with_Attentive_Multi-View_Learning. (accessed: 02.05.2024).
- [4] Gohlamy et al. *Why 70/30 or 80/20 Relation Between Training and Testing Sets: A Pedagogical Explanation*. 2024. URL: https://scholarworks.utep.edu/cs_techrep/1209/. (accessed: 28.05.2024).
- [5] Leysen et al. *What Are Filter Bubbles Really? A Review of the Conceptual and Empirical Work*. 2022. URL: https://www.researchgate.net/publication/362968157_What_Are_Filter_Bubbles_Really_A_Review_of_the_Conceptual_and_Empirical_Work. (accessed: 16.05.2024).
- [6] Mingxiao et al. *Neural News Recommendation with Long- and Short-term User Representations*. 2020. URL: <https://paperswithcode.com/paper/neural-news-recommendation-with-long-and>. (accessed: 02.05.2024).
- [7] Wu et al. *MIND: Microsoft News Dataset*. 2020. URL: <https://msnews.github.io/>. (accessed: 02.05.2024).

- [8] *Are younger generations moving away from traditional news sources?* | Deloitte. URL: <https://www2.deloitte.com/dk/da/pages/technology-media-and-telecommunications/topics/digital-consumer-trends/are-younger-generations-moving-away-from-traditional-news-sources.html>. (accessed: 14.05.2024).
- [9] *Axios*. URL: <https://axios-http.com/>. (accessed: 15.04.2024).
- [10] Mayur Badole. *A Comprehensive Guide on Recommendation Engines and Implementation*. 2024. URL: <https://www.analyticsvidhya.com/blog/2022/03/a-comprehensive-guide-on-recommendation-engines-and-implementation/>. (accessed: 26.02.2024).
- [11] Ekstra Bladet. *Codebench*. 2024. URL: https://www.codabench.org/competitions/2469/?secret_key=98314b2c-9237-471e-905c-2a88bf6a1d8a#/pages-tab. (accessed: 15.05.2024).
- [12] Ekstra Bladet. *RecSys Challenge 2024*. 2024. URL: <https://recsys.eb.dk/>. (accessed: 23.05.2024).
- [13] Salesforce Marketing Cloud. *Predictive Intelligence Benchmark Report*. 2014. URL: <https://brandcdn.exacttarget.com/sites/exacttarget/files/deliverables/etmc-predictiveintelligencebenchmarkreport.pdf>. (accessed: 26.02.2024).
- [14] *Daniel Pleus* | LinkedIn. URL: <https://www.linkedin.com/in/daniel-pleus/?originalSubdomain=se>. (accessed: 15.04.2024).
- [15] Aparna Dhinakaran. *Demystifying NDCG*. 2023. URL: <https://towardsdatascience.com/demystifying-ndcg-bee3be58cfe0>. (accessed: 15.05.2024).
- [16] Jeffrey Erickson. *What Is JSON?* 2024. URL: <https://www.oracle.com/database/what-is-json/>. (accessed: 27.05.2024).
- [17] *FastAPI*. URL: <https://fastapi.tiangolo.com/>. (accessed: 15.04.2024).

- [18] GeeksForGeeks. *Software Quality Assurance – Software Engineering*. 2024. URL: <https://www.geeksforgeeks.org/software-engineering-software-quality-assurance/>. (accessed: 06.05.2024).
- [19] GitHub. *GitHub Copilot - The world's most widely adopted AI developer tool*. 2024. URL: <https://github.com/features/copilot>. (accessed: 26.02.2024).
- [20] GCF Global. *Digital Media Literacy: How filter bubbles isolate you*. 2022. URL: <https://edu.gcfglobal.org/en/digital-media-literacy/how-filter-bubbles-isolate-you/1/#>. (accessed: 16.05.2024).
- [21] Google. *Classification: ROC Curve and AUC*. 2024. URL: <https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc>. (accessed: 15.05.2024).
- [22] *Impact of learning rate on a model - GeeksforGeeks*. URL: <https://www.geeksforgeeks.org/impact-of-learning-rate-on-a-model/>. (accessed: 10.05.2024).
- [23] *Introduction to the TikTok recommendation system | TikTok*. URL: <https://www.tiktok.com/transparency/en-us/recommendation-system/>. (accessed: 16.05.2024).
- [24] Luxuan Wang Jacob Liedke. *News Platform Fact Sheet | Pew Research Center*. Nov. 2023. URL: <https://www.pewresearch.org/journalism/fact-sheet/news-platform-fact-sheet/?tabItem=5a0b8b87-38bc-42d6-ba8d-2e666200e534>. (accessed: 16.05.2024).
- [25] Héla Ben Khalfallah. *How Can We Measure Our Software's Modularity and Dependencies?* 2020. URL: <https://betterprogramming.pub/inside-software-modularity-and-related-metrics-2e5af2b447dc>. (accessed: 06.05.2024).
- [26] Benjamin Kille et al. "The plista dataset". In: *Proceedings of the 2013 international news recommender systems workshop and challenge*. 2013, pp. 16–23.

- [27] *Layered Architecture in Computer Networks - GeeksforGeeks*. URL: <https://www.geeksforgeeks.org/layered-architecture-in-computer-networks/>. (accessed: 22.05.2024).
- [28] Leaderboard. *RecSys Challenge 2024*. 2024. URL: <https://recsys.eb.dk/#leaderboard>. (accessed: 27.05.2024).
- [29] Universidad Carlos III de Madrid. *The impact of targeted Facebook advertising on the 2016 United States presidential election*. 2018. URL: <https://www.sciencedaily.com/releases/2018/11/181119155940.htm>. (accessed: 16.05.2024).
- [30] *Matplotlib — Visualization with Python*. URL: <https://matplotlib.org/>. (accessed: 09.05.2024).
- [31] *MIND recommender from scratch*. URL: <https://www.kaggle.com/code/danielpheus/mind-recommender-from-scratch/notebook>. (accessed: 15.04.2024).
- [32] MÖBIUS. *MIND: Microsoft News Recommendation Dataset*. URL: <https://www.kaggle.com/datasets/arashnic/mind-news-dataset/data>. (accessed: 15.04.2024).
- [33] *News Portal User Interactions by Globo.com*. URL: <https://www.kaggle.com/datasets/gspmoreira/news-portal-user-interactions-by-globocom>. (accessed: 22.05.2024).
- [34] Nvidia. *Recommendation System*. 2024. URL: <https://www.nvidia.com/en-us/glossary/recommendation-system/>. (accessed: 26.02.2024).
- [35] OpenAI. *Introducing ChatGPT*. 2022. URL: <https://openai.com/blog/chatgpt>. (accessed: 06.05.2024).
- [36] Roman Panarin. *Recommender System Using Machine Learning*. 2024. URL: <https://maddevs.io/blog/recommender-system-using-machine-learning/>. (accessed: 26.02.2024).

- [37] Justin Basilico & Yves Raimond. *Deep Learning for Recommender Systems*. 2024. URL: <https://www.nvidia.com/en-us/glossary/recommendation-system/>. (accessed: 26.02.2024).
- [38] Danske Medier Research. *Toplisten*. 2024. URL: <https://www.fdim.dk/statistik/internet/toplisten>. (accessed: 26.02.2024).
- [39] Chan Rosalie. *The impact of targeted Facebook advertising on the 2016 United States presidential election*. 2019. URL: <https://www.businessinsider.com/cambridge-analytica-whistleblower-christopher-wylie-facebook-data-2019-10>. (accessed: 16.05.2024).
- [40] Data Scientist. *The importance of Cross Validation*. 2024. URL: <https://datascientest.com/en/the-importance-of-cross-validation>. (accessed: 28.05.2024).
- [41] *Shape Up: Stop Running in Circles and Ship Work that Matters*. URL: <https://basecamp.com/shapeup>. (accessed: 20.05.2024).
- [42] Twitter Support. *About Community Notes on X*. 2024. URL: <https://help.twitter.com/en/using-x/community-notes>. (accessed: 16.05.2024).
- [43] Amazon Web Services Team. *Factorization Machines Algorithm*. 2024. URL: <https://docs.aws.amazon.com/sagemaker/latest/dg/fact-machines.html#>. (accessed: 06.05.2024).
- [44] CodeScene Team. *CodeScene - Next generation code analysis*. 2024. URL: <https://codescene.com/>. (accessed: 26.05.2024).
- [45] Databricks Team. *What is MLOps?* 2024. URL: <https://www.databricks.com/glossary/mlops>. (accessed: 26.05.2024).
- [46] FastAPI Team. *FastAPI Documentation*. 2024. URL: <https://fastapi.tiangolo.com/>. (accessed: 28.05.2024).
- [47] Flask Team. *Flask Documentation*. 2024. URL: <https://flask.palletsprojects.com/en/3.0.x/>. (accessed: 28.05.2024).

- [48] Kubeflow Team. *Kubeflow Documentation*. 2024. URL: <https://www.kubeflow.org/>. (accessed: 28.05.2024).
- [49] LightFM Team. *LightFM Deep Dive*. 2022. URL: https://github.com/recommenders-team/recommenders/blob/main/examples/02_model_collaborative_filtering/lightfm_deep_dive.ipynb. (accessed: 06.05.2024).
- [50] LightFM Team. *LightFM FAQ*. 2022. URL: <https://making.lyst.com/lightfm/docs/index.html>. (accessed: 06.05.2024).
- [51] MLFlow Team. *MLFlow Documentation*. 2024. URL: <https://mlflow.org/>. (accessed: 28.05.2024).
- [52] Recommenders Team. *DKN : Deep Knowledge-Aware Network for News Recommendation*. 2020. URL: https://github.com/recommenders-team/recommenders/blob/main/examples/00_quick_start/dkn_MIND.ipynb. (accessed: 07.05.2024).
- [53] The NYT Open Team. *Machine Learning and Reader Input Help Us Recommend Articles | NYT Open*. Jan. 2021. URL: <https://open.nytimes.com/we-recommend-articles-with-a-little-help-from-our-friends-machine-learning-and-reader-input-e17e85d6cf04>. (accessed: 17.05.2024).
- [54] *The Guardian - Live World News on the App Store*. URL: <https://apps.apple.com/us/app/the-guardian-live-world-news/id409128287>. (accessed: 8.05.2024).
- [55] *Tinder: Chat, Dating & Friends on the App Store*. URL: <https://apps.apple.com/us/app/tinder-chat-dating-friends/id547702041>. (accessed: 08.05.2024).
- [56] *Twitter's Recommendation Algorithm*. URL: https://blog.x.com/engineering/en_us/topics/open-source/2023/twitter-recommendation-algorithm. (accessed: 16.05.2024).

- [57] *Understanding Layered Architecture: A Comprehensive Guide* | by Satyendra Jaiswal | Medium. URL: <https://medium.com/@satyendra.jaiswal/understanding-layered-architecture-a-comprehensive-guide-4c2eee374d18>. (accessed: 22.05.2024).
- [58] Rui Vasconcelos. *A guide to ML model serving*. 2024. URL: <https://ubuntu.com/blog/guide-to-ml-model-serving>. (accessed: 02.05.2024).
- [59] Dina Vozab. *Tracking the Relationship Between Media Literacy and Political Participation Across Different Generations*. 2023. URL: <https://hrcak.srce.hr/308325>. (accessed: 16.05.2024).
- [60] *What is the Meaning of Latent Features?* - GeeksforGeeks. URL: <https://www.geeksforgeeks.org/what-is-the-meaning-of-latent-features/>. (accessed: 10.05.2024).
- [61] Wikipedia. *Cold start (recommender systems)*. 2024. URL: [https://en.wikipedia.org/wiki/Cold_start_\(recommender_systems\)](https://en.wikipedia.org/wiki/Cold_start_(recommender_systems)). (accessed: 28.05.2024).
- [62] Summer Worsley. *What is SQL?* 2022. URL: <https://www.datacamp.com/blog/all-about-sql-the-essential-language-for-database-management>. (accessed: 27.05.2024).
- [63] Fangzhao Wu et al. *Mind: A large-scale dataset for news recommendation*. 2020.

Appendices

Appendix A

Noter fra mødet 16/02

- Vi har ikke mulighed for at få adgang til deres recommender system
 - Dog kan vi få metadata
 - Heller ikke mulighed for at tilgå API
- EB mener vi måske burde begrænse os
 - Hvis vi både laver machine learning og mobil app kan det blive et meget stort project
- Andre features man kunne overveje
 - Ikke kun anbefale artikler, men også anbefale de næste artikler efter den første
- Ting vi må få adgang til
 - Vi kan bede om et dataset og så skal vi skrive en NDA
 - Vi kan ikke få adgang til deres recommender system, men vi kan spørge ind til det
- Info om deres recommender system
 - Brugeren behøver ikke være logget ind for at få anbefalet artikler
 - Onboarding starter allerede på forsiden
- Deres ønsker
 - Named entity recognition
 - Anbefal artikler ud fra navne fra de tidligere artikler, feks
 - Navne
 - Områder
 - De har allerede den data, de bruger det bare ikke
 - Hvordan evaluerer vi det?
 - Få en masse venner til at prøve det gamle produkt og det nye
 - De vil gerne ændre måden de anbefaler artikler på deres side
 - Bruge generative AI til at give en forklaring på specifikke keywords
- Muligheder
 - Vi kan ansøge om artikler, interval og så modtager vi resultater med artiklen og NER + sentiment data
 - Ellers kan vi træne vores egen model

Sv: Update from LAB

Kristoffer Hartwig <Kristoffer.Hartwig@jppol.dk>

fr 22-03-2024 10:02

Til: Anders Mazen Youssef <amyo21@student.aau.dk>; Bence Szabó <bszabo21@student.aau.dk>;

Cc: Louise Foldøy Steffens <lfst21@student.aau.dk>;

Hej LAB,

Det er læst og godkendt herfra.

Tak for mødet i går, og vi høres ved!

Kristoffer Hartwig

Produktudvikler

Mail: kristoffer.hartwig@eb.dk

Mobil: +45 60 14 18 65

Fra: Anders Mazen Youssef <amyo21@student.aau.dk>

Sendt: 14. marts 2024 12:55

Til: Kristoffer Hartwig <Kristoffer.Hartwig@jppol.dk>; Bence Szabó <bszabo21@student.aau.dk>

Cc: Louise Foldøy Steffens <lfst21@student.aau.dk>

Emne: Sv: Update from LAB

Hej Kristoffer,

Der er sket en del fremskridt på produktet og vi ville høre om du havde mulighed for et møde på tirsdag? Vi tænker at forberede en powerpoint med de vigtigste punkter og præsentere det. Vi kan både mødes fysisk eller online, hvad end passer dig bedst. 😊

Derudover, så ville vi også spørge om det var muligt for jer at læse vores noter vi skrev ned under vores første møde og bekræfte om der er nogle misforståelser eller ej. Grunden til det er så vi kan referere til det i vores rapport, i stedet for at lave et helt nyt interview som vi skal optage og transskribere.

Tak på forhånd!

- Anders, Bence & Louise

Fra: Kristoffer Hartwig <Kristoffer.Hartwig@jppol.dk>

Sendt: 14. marts 2024 09:19:53

Til: Bence Szabó

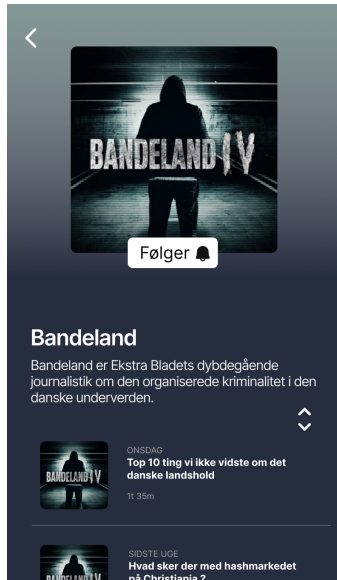
Cc: Anders Mazen Youssef; Louise Foldøy Steffens

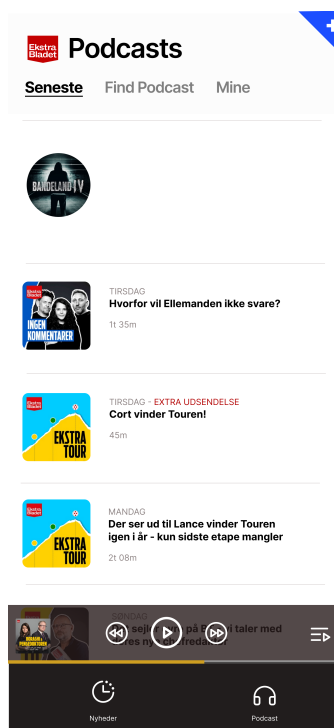
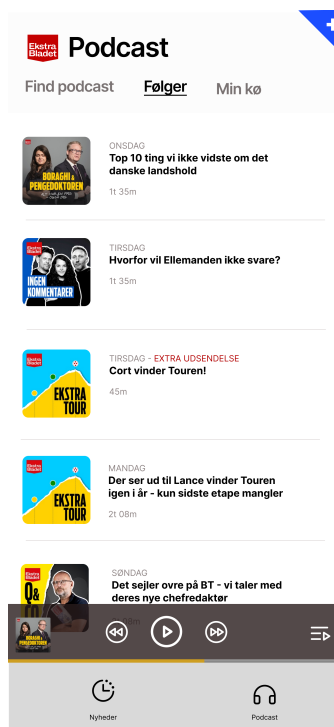
Emne: Sv: Update from LAB

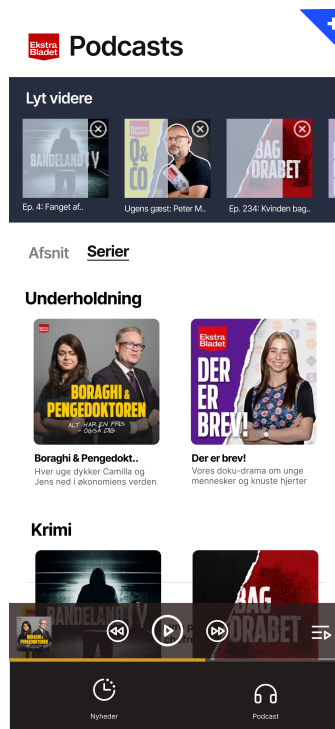
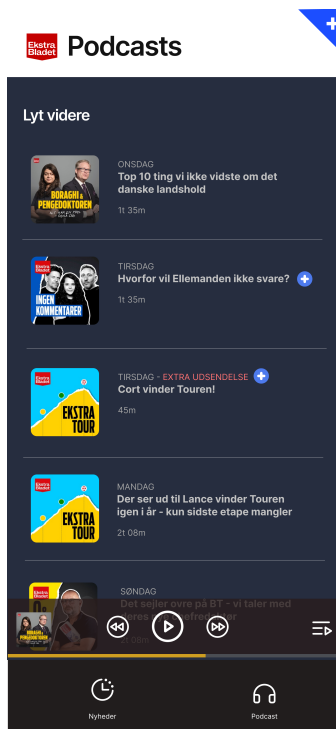
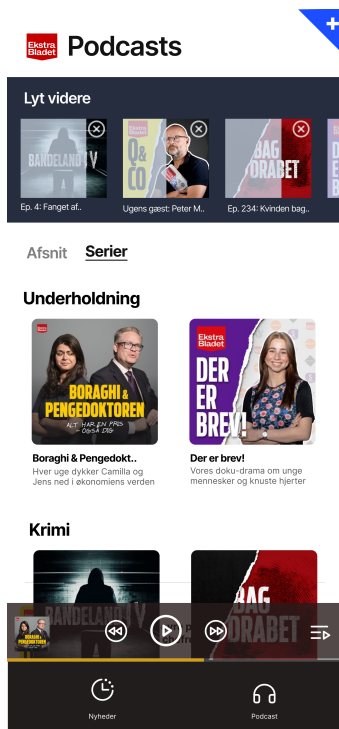
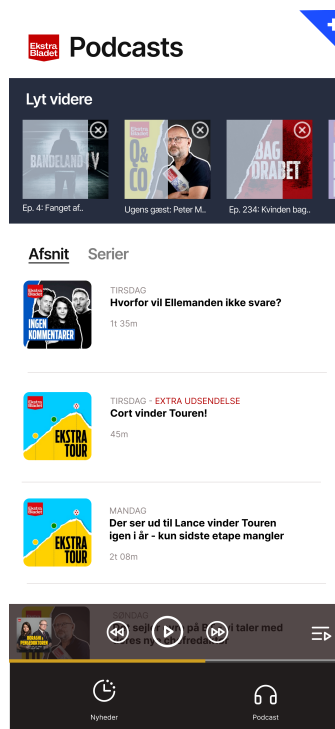
Hej igen!


Appendix B

These are all the images from the figma provided by Ekstra Bladet. These describe the design they wished for in the application.












Podcasts

Lyt videre


Ep. 4: Fanget af...

Ugens gæst: Peter M...

Ep. 234: Kvinden bag...

Afsnit


Serier



TIRSDAG

Hver for sig til Ellemanden ikke svare?


1t 15m



TIRSDAG - EXTRA UDSEDELSE

Cort vinder Touren!




45m




MANDAG

Der ser ud til Lance vinder Touren igen i år - kun sidste etape mangler

2t 08m




Nyheder

Gad

Podcast


[illegible]



Podcasts


Afsnit Serie

Underholdning



Boraghi & Pengedoktoren
Alt i livet på lyd - 200 af 250


Boraghi & Pengedokt..
Hver uge dykker Camilla og Jens ned i økonomiens verden



Der er brev!
Vores dokumentar om unge mennesker og knuste hjerner


Der er brev!
Vores dokumentar om unge mennesker og knuste hjerner

Krimi



BAGLANDET IV


Boraghi & Pengedokt..
Hver uge dykker Camilla og Jens ned i økonomiens verden



BAGDRABET


Boraghi & Pengedokt..
Hver uge dykker Camilla og Jens ned i økonomiens verden

Sport



Nyheder

Nyheder



Podcast

Podcast

Appendix c

SV: LAB Update + møde

Kristoffer Hartwig <Kristoffer.Hartwig@jppol.dk>

to 02-05-2024 08:47

Til: Louise Foldø Steffens <lfst21@student.aau.dk>; Anders Mazen Youssef <amyo21@student.aau.dk>; Bence Szabó <bszabo21@student.aau.dk>;

Så er der kommet svar fra den gode Jesper Rix:

"Vi vil gerne undersøge mulighederne for at genskabe (eller dokumentere) den fysiske/cloud model-serving setup (MLOps), som vi skal tage højde for, hvis appen når hele vejen til produktion og skal offentliggøres for brugerne.

Vi er ved at ændre vores infrastruktur til kubernetes og Argo Workflow, med en inference server, som skal hoste vores recommender systemer alå Pytorch Serve eller Nvidia tritonMen det nuværende setup er noget vi selv har bygget med AWS CDK, hvor modellerne trænes i AWS Batch og er i produktion, som Python services med Flask eller FastAPI i docker containere, som køres i AWS ECS. Selve modellerne bliver gemt som artefakter på S3 og derefter hentet ind i de førnævnte services.

Så har vi en Go applikation, som bl.a. står routing mellem modeller og for at sætte A/B test op imellem modeller eller forskellige kombinationer osv.Er det svar nok, ellers kan vi uddybe yderligere.

Mere specifikt indeholder det:

- *Hvordan er model training / retrain / partial retrain konfigureret? (Er det 100% manual proces, eller er der fx automatisk retrain efter en bestemt mængde ny data er kommet ind i databasen?)*

Vores collaborative filtering modeller træner hver 2. time da de ellers har "cold start" problemer da ekstra bladet udgiver mange artikler på 2 timer. De content baserede modeller trænes en gang i døgnet og er mere robuste og kan godt håndtere "cold start" vi har kørt tests hvor vi *kun* trænede én gang på en uge vs. hver dag hvor der var lidt bedre performance ved at træne hver dag.

Vi har endnu ikke kigget på metrik styret gentræning af modellerne.

- *Hvordan bliver modellen deployed / udskiftet i deployment?*

AWS ECS servicen bliver "re-deploy'et" hvilket tvinger containeren til at hente det nyeste artefakt ned.

- *Hvordan bliver modellen monitoreret? (Opsætning af overvågningsværktøjer og advarsler for at opdage problemer som datadrift, modelforringelse eller uventet adfærd)*

Vi opsamler en masse data omkring modellernes performance, som vi har kigget på afledte effekter i nyhedsstrømmen i, men det er ikke noget vi på nuværende tidspunkt bruger til monitorering, men det har vi planer om.

Det eneste vi kigger på er salg eller CTR.

- *Generelt om pipeline - Hvordan kommer ny kode / ny model fra developer til production?"*

Her bruger vi CI/CD principper, så når en pull request på github merges til master/main sker test+deployment automatisk.

--

Kristoffer Hartwig
Produktudvikler

Mail: kristoffer.hartwig@eb.dk

Mobil: [+45 60 14 18 65](tel:+4560141865)

Fra: Kristoffer Hartwig <Kristoffer.Hartwig@jppol.dk>

Dato: tirsdag, 30. april 2024 kl. 09.28

Til: Louise Foldøy Steffens <lfst21@student.aau.dk>, Anders Mazen Youssef <amyo21@student.aau.dk>, Bence Szabó <bszabo21@student.aau.dk>

Emne: SV: LAB Update + møde

Hej alle,

Beklager langsommeligheden fra min side igen – det er som om der er sker noget nyt og uforudset heroppe hver dag.

Spørgsmålet er sendt videre til Jesper Rix, som er af lederne i AI-enheden her i huset.

Ift. Figma, så sender jeg hele filen her:

 [EB App.fig](#)

Det er lettere rodet, men kig i "Assets", "Newsreader" og endelig "EB Ekstra - Next Gen" for det i leder efter.

Vi ses!

/ Kristoffer

--

Kristoffer Hartwig
Produktudvikler

Mail: kristoffer.hartwig@eb.dk

Mobil: [+45 60 14 18 65](tel:+4560141865)

Fra: Bence Szabó <bszabo21@student.aau.dk>

Dato: torsdag, 25. april 2024 kl. 13.58

Til: Kristoffer Hartwig <Kristoffer.Hartwig@jppol.dk>

Emne: Re: LAB Update + møde

Hej, og tak for mødet i dag!)

Kan du muligvis sende Figma filerne til timeline og evt. andre nye idéer til appen?

Og her er vores tidligere MLOps spørgsmål med lidt mere beskrivelse:

Vi vil gerne undersøge mulighederne for at genskabe (eller dokumentere) den fysiske/cloud model-serving setup (MLOps), som vi skal tage højde for, hvis appen når hele vejen til produktion og skal offentliggøres for brugerne.

Mere specifikt indeholder det:

- Hvordan er model training / retrain / partial retrain konfigureret? (Er det 100% manual proces, eller er der fx automatisk retrain efter en bestemt mængde ny data er kommet ind i databasen?)

- Hvordan bliver modellen deployed / udskiftet i deployment?

- Hvordan bliver modellen monitoreret? (Opsætning af overvågningsværktøjer og advarsler for at opdage problemer som datadrift, modelforringelse eller uventet adfærd)

- Generelt om pipeline - Hvordan kommer ny kode / ny model fra developer til production?

Mvh,

LAB

From: Kristoffer Hartwig <Kristoffer.Hartwig@jppol.dk>

Sent: Monday, April 22, 2024 9:44:41 PM

To: Bence Szabó

Subject: SV: LAB Update + møde

Hej LAB,

9.25 er fint. Så sørger jeg for et møderum til os.

/ kristoffer

Fra: Bence Szabó <bszabo21@student.aau.dk>

Dato: mandag, 22. april 2024 kl. 17.24

Til: Kristoffer Hartwig <Kristoffer.Hartwig@jppol.dk>

Cc: Louise Foldø Steffens <lfst21@student.aau.dk>, Anders Mazen Youssef <amyo21@student.aau.dk>

Emne: Re: LAB Update + møde

Torsdag formiddag kl 09:30 passer fint. :) Og det passer stadig fysisk hos jer? Hvis ja, hvad tid skal vi være ved lobbyen?

Mvh,

LAB

From: Kristoffer Hartwig <Kristoffer.Hartwig@jppol.dk>

Sent: Monday, April 22, 2024 9:20:46 AM

To: Bence Szabó

Cc: Louise Foldø Steffens; Anders Mazen Youssef

Subject: SV: LAB Update + møde

Torsdag formiddag kunne være en god mulighed. Har dog noget der starter 11. Så hvis det er f.eks. 9:30 eller 10?

Fra: Bence Szabó <bszabo21@student.aau.dk>

Dato: søndag, 21. april 2024 kl. 12.28

Til: Kristoffer Hartwig <Kristoffer.Hartwig@jppol.dk>

Cc: Louise Foldø Steffens <lfst21@student.aau.dk>, Anders Mazen Youssef <amyo21@student.aau.dk>

Emne: Re: LAB Update + møde

Desværre passer tirsdag ikke helt optimalt for os. Hvad med at mødes fysisk onsdag kl. 10 eller torsdag formiddag?

Mvh,

LAB

From: Kristoffer Hartwig <Kristoffer.Hartwig@jppol.dk>

Sent: Thursday, April 18, 2024 4:08:56 PM

To: Bence Szabó

Cc: Louise Foldøy Steffens; Anders Mazen Youssef

Subject: SV: LAB Update + møde

Hej LAB,

Godt at høre fra jer, og fedt med 3. sprint.

Sig til hvad der passer jer bedst ift at mødes i næste uge? Måske Tirsdag? (online / fysisk er op til jer).

Ift de to specifikke spørgsmål, skal jeg sørge for at undersøge 1, og sørge for at bringe 2 videre til Johannes. Tak!

Vi høres ved,

Kristoffer

Fra: Bence Szabó <bszabo21@student.aau.dk>

Dato: torsdag, 18. april 2024 kl. 14.36

Til: Kristoffer Hartwig <Kristoffer.Hartwig@jppol.dk>

Cc: Louise Foldøy Steffens <lfst21@student.aau.dk>, Anders Mazen Youssef <amyo21@student.aau.dk>

Emne: LAB Update + møde

Hej Kristoffer!

Vi tænker at det kunne være en god ide at holde et opdateringsmøde, siden vi bliver færdige med vores 3. sprint her på fredag. Hvilken dag ville passe bedst med dig, og vil du helst tage det over Teams eller fysisk?

Udover mødet, har vi lige et spørgsmål angående MLOps, og en vigtig observation til EBNeRD datasættet.

Vi har et spørgsmål specifikt til MLOps, som kunne være super brugbart i vores rapport:

- Vil vi gerne undersøge mulighederne for at genskabe (eller i det mindste dokumentere) den fysiske/cloud model-serving setup (MLOps), som vi skal tage højde for, hvis appen når hele vejen til produktion og skal offentliggøres for brugerne. Vil det være muligt for dig at dele information om disse ting?

Vi har opdaget en mulig exploit man kunne bruge til at snyde i EBNeRD konkurrencen:

- Under data-preparation fasen har vi lagt mærke til at det er muligt at extracte data fra validation dataset som resulterer i en overlap imellem training og validation clicks/noclicks. Dette skyldes at der er en del overlap imellem clicked articles i training history og validation history. Vi har valgt at fjerne dem fra vores validering for at sikre korrekte resultater, men hvis man ikke gør det, så vil omkring $\frac{1}{3}$ af ens validation AUC score være lige med training AUC score, som plejer at være markant højere end validation.

- Kort sagt, er det selvfølgelig meget nemt at forudsige at en user vil læse en artikel, hvis der allerede er data på at user'en har læst den specifikke artikel. Hvis vores observationer er korrekte, så kan det have stor betydning for hvor gode EBNeRD konkurrencens bedste modeller i virkeligheden er.

Mvh,
Lab

Appendix d

Sv: Opdatering fra LAB

Kristoffer Hartwig <Kristoffer.Hartwig@jppol.dk>

fr 17-05-2024 13:13

Til: Bence Szabó <bszabo21@student.aau.dk>;

Cc: Louise Foldø Steffens <lfst21@student.aau.dk>; Anders Mazon Youssef <amyo21@student.aau.dk>;

- Hvis vi skulle arbejde videre med appen / den måde machine learning er implementeret på, hvad synes du ville være mest relevant / interessant at fokusere på?

Med ét ord : **Nyhedsoplevelsen** - altså både hvordan nyheder bliver udvalgt til brugeren, og hvordan vi kan berige artiklerne med kontekst.

For at give lidt baggrund: Med appen kommer der til at være fokus på hvordan man får folk i alderen 18-28 til at blive mere engagerede i nyheder. Kan vi tilbyde dem (jer!) en værdi som man ikke får andre steder, og som man i én eller anden grad vil være villig til at betale for?

Grunden til betaling er vigtig, er at jeg rigtig gerne vil dreje produktet væk fra at være optimeret til klik/sidevisninger/annoncering, og hen imod et dedikeret fokus på nyhedsoplevelsen.

Som det er nu, ender man hurtigt i click-bait, sensationer og doomscrolling. Det mener jeg er ærgeligt, fordi EB faktisk har en masse magtkritisk, god, sjov og vigtig journalistik at byde på.

Jeg antager, at vi kunne bruge recommendere til at give kontekst og indsigt i de enkelte artikler - og til at tilbyde et nyhedsfeed som er bygget op om brugeren, og ikke om en nyhedsredaktion.

Om man vil starte med feedet, eller længere nede i artiklerne, er en smagssag som jeg ser det. Ville nok starte med feedet.

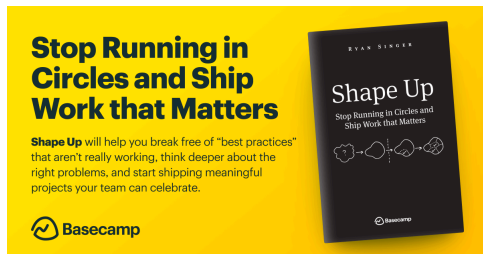
- Hvad synes du om kommunikationen/samarbejdesprocessen igennem projektet?

Jeg synes klart i har leveret over forventning, også når jeg sammenligner med tidligere erfaringer i samme felt. I har haft styr på jeres ting. I har været gode til at præsentere jeres fremgang. Jeg synes det skinner igennem, at i er en god gruppe, som både er sympatiske, kompetente og seriøse. Om noget, så skal det være min beklagelse, at det nogle gange har taget flere dage at få svaret på jeres mails, eller skaffet kontakt til de rigtige mennesker her i huset.

Som product owner, føler du, at din feedback har haft en stor nok indflydelse på produktet?

Ja helt klart.

Her på EB benytter vi os den product management metodik, der hedder "[Shape Up](#)" og den vil sige at jeg ikke må være alt for "præcis" i de opgaver som udviklerne får. Hvis vi skulle følge den, så jeg bare have givet jer et skarpt problem, og konturerne eller formen på en løsning, som i så ville bygge. Så i den sammenhæng har jeg nærmest haft for meget indflydelse :)



Shape Up: Stop Running in Circles and Ship Work that Matters

Shape Up will help you break free of “best practices” that aren’t really working, think deeper about the right problems, and start shipping meaningful projects your team can celebrate.

basecamp.com

Håber det kan bruges - og sig til når vi skal følge op + vi skal have fundet en fredag, hvor i kommer ind i præsentationer!

God weekend,
Kristoffer

Kristoffer Hartwig
Produktudvikler

Mail: kristoffer.hartwig@eb.dk
Mobil: +45 60 14 18 65

Fra: Kristoffer Hartwig <Kristoffer.Hartwig@jppol.dk>
Sendt: 17. maj 2024 08:11
Til: Bence Szabó <bszabo21@student.aau.dk>
Cc: Louise Foldøy Steffens <lfst21@student.aau.dk>; Anders Mazen Youssef <amyo21@student.aau.dk>
Emne: Re: Opdatering fra LAB

Hej alle tre,

Tak for mail og godt at se at i (i hvert fald Bence) havde tid til at nyde solen forleden 😊

Jeg kommer tilbage med svar senere i dag. Der har været lidt meget run på heroppe og vil gerne lige tænke ordentligt inden jeg svarer.

I hører fra mig!

Sent from [Outlook for iOS](#)

From: Bence Szabó <bszabo21@student.aau.dk>
Sent: Wednesday, May 15, 2024 10:52:57 AM
To: Kristoffer Hartwig <Kristoffer.Hartwig@jppol.dk>
Cc: Louise Foldøy Steffens <lfst21@student.aau.dk>; Anders Mazen Youssef <amyo21@student.aau.dk>
Subject: Opdatering fra LAB

Hej Kristoffer! :)

Vi er ved at afslutte vores sidste sprint, så vi har en lille statusopdatering:

App

- Vi er blevet færdige med den swipe-able article card komponent, og har efter din feedback tilføjet et overlay som forsvinder når man rører ved den (se screenshots).
- Vi har fixet en del minor bugs i appen, og har opdateret alle dependencies.

Model

- Det går ret godt med model training, vi får solid 0.9 training AUC and 0.89 testing AUC med de tweaks vi har lavet (se vedhæftet screenshot).
- Når vi har trænet en model, så gemmer vi en package med configs, evaluation data, og selve modellen. Dette betyder, at man kan super nemt få et overblik over modellens konfiguration og performance, og fortsætte med at træne modellen.

Testing og QA

- Vi har udvidet vores frontend og backend testing for at sikre at alting virker som det skal.
- Når vi er færdige med at skrive tests, vil vi rydde både front-og backend op mht. modularitet, og vil køre CodeScene statisk kodeanalyse på dem for at sikre at vores kode er klar og læselig.

Vi har også et par spørgsmål som er vigtig ift. rapporten:

- Hvis vi skulle arbejde videre med appen / den måde machine learning er implementeret på, hvad synes du ville være mest relevant / interessant at fokusere på?
- Hvad synes du om kommunikationen/samarbejdesprocessen igennem projektet? Som product owner, føler du, at din feedback har haft en stor nok indflydelse på produktet?

Når vi er helt færdige med kode-delen af projektet, vil vi også gerne give en afsluttende præsentation, hvor vi kan evt. diskutere de ovenstående spørgsmål, og snakke om hvad du egentlig synes om det færdige produkt. Vi kommer med et forslag til dato asap.

Mvh,
LAB

Appendix e

AI-powered tools

During the project, GitHub Copilot and ChatGPT 3.5 [35] are used to assist the team's workflow and overall efficiency. GitHub Copilot [19] is a context-aware code completion tool used to boost coding speed. ChatGPT is primarily used to give suggestions for fixing relatively simple technical issues and as a helping hand in finding the right answers to questions that may arise during research. ChatGPT is not treated as a reliable source.

Appendix f

This section contains details about the rest of the completed PBIs from Sprint 1, which are:

- S1F1: As a user, I want to have a non-personalised news feed.
- S1F3: As a user, I want a header on my personalized feed.
- S1F5: As a user, I want to be able to see if I'm a plus member or not.
- S1F6: As a user, I want to navigate from the news feed to an article by tapping on it.
- S1F8: As a user, I want to see a loading screen with Ekstra Bladet's logo when opening the application.
- S1F9: As a user, I want to see the correct fonts when using the app.
- S1F10: As a user, I want to have paragraphs in the article.
- S1F12: As a user, I want to see breaking news with special styling.

S1F1: As a user, I want to have a non-personalised news feed.

This PBI implements getting unfiltered news-feed which only shows the latest news available. The personalised news-feed component has been altered to be able to support this PBI. This means that all the styling and functionality is exactly the same, and only the data is different.

S1F3: As a user, I want a header on my personalized feed.

This PBI addresses another component from the design provided by Ekstra Bladet.

9 The component for this PBI is the top bar of the landing page of the application, it contains the title of each sub-view the user can access while also changing appearance depending on the sub-view pressed by the user. There is also a logo present next to the main header.

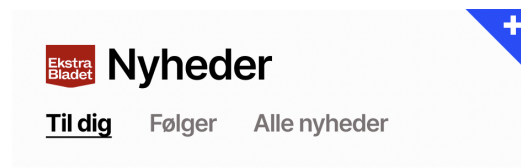


Figure 10: Top bar component

The logic for switching between the different sub-menus has also been implemented, but other than the personalised newsfeed screen, they redirect to empty screens.

S1F5: As a user, I want to be able to see if I'm a plus member or not.

This PBI focuses on implementing a component used to indicate whether the user is subscribed to Ekstra Bladets premium subscription service. The component is either blue if the user is subscribed, or grey if they are not. It is created as a component so it can easily be implemented on any screen by the developer. Currently, it is present on all screens following the product owner's request.



Figure 11: Article screen with the subscription component implemented

S1F6: As a user, I want to navigate from the news feed to an article by tapping on it.

This PBI addresses the previously missing feature of navigating to an article when pressing the corresponding news card. This was done by using a function to navigate the screen from the landing page to the article screen while passing

the article object onto the article screen.

Listing 1: Navigating to the article screen from the news feed

```
1 const handleOnPress = () => {  
2   navigation.navigate('Article', { article: article });  
3 };
```

Once in the article, the back-arrow button returns the user to the news feed screen using the *navigation.goBack()* function.

S1F8: As a user, I want to see a splash screen with Ekstra Bladet's logo when opening the application.

This PBI is relevant in terms of presentation, and it was also fairly simple to implement. DR Nyheder, TV2, Twitter, Facebook, and almost all other popular apps use it so the screen contents loading are not visible to the user, and is generally also considered a good design practice. The splash screen is shown when the application is opened and presents the user with a logo fading in while the app is loading, and fades out when the app is ready to be used by the user. Note that an extra timer is added for a better user experience, otherwise the splash screen would flash.



Figure 12: Splash screen shown while the app is loading

Afterwards, the screen smoothly navigates to the next screen, the news feed showing the user their predicted articles.

S1F9: As a user, I want to see the correct fonts when using the app.

This PBI addresses the lack of specified fonts in the application and aims to implement the same fonts used in the Figma provided by Ekstra Bladet. This was done by adding a function that loads the relevant fonts placed within the "assets/fonts" directory whenever the application starts. However, since the design

provided for the article screen did not mention which fonts were used, the fonts for the said screen will be addressed in a later PBI when more information about its design has been given. The group notes this and consults the product owner at the following meeting.

Listing 2: Function which loads the specified fonts from the assets/fonts directory

```
1 const [fontsLoaded] = useFonts({
2   "WorkSans-Regular": require("./assets/fonts/WorkSans-Regular.ttf"),
3   "WorkSans-Medium": require("./assets/fonts/WorkSans-Medium.ttf"),
4   "Karla-Regular": require("./assets/fonts/Karla-Regular.ttf"),
5   "Karla-Medium": require("./assets/fonts/Karla-Medium.ttf"),
6   "InterTight-SemiBold": require("./assets/fonts/InterTight-SemiBold.ttf"),
7   "InterTight-Bold": require("./assets/fonts/InterTight-Bold.ttf"),
8 });
9
10 if (!fontsLoaded) {
11   return <Text>Loading...</Text>;
12 }
```

S1F10: As a user, I want to have paragraphs in the article

This PBI addresses the issue of how the body of an article is rendered. Before the implementation of this PBI, the raw HTML code would be displayed to the user, and it would not have any separation of paragraphs. This makes the body text of an article incredibly difficult to read and hurts the user experience.

Fixing this was done by placing the body text within an array which splits each paragraph by every HTML keyword for creating a new line, which is \n. Afterwards, the array is mapped through and rendered with a new line after each paragraph.

Listing 3: Function which loads the specified fonts from the assets/fonts directory

```
1 const paragraphs = article.body.split('\n');
```



```
2
3 const renderedParagraphs = paragraphs.map((paragraph, index) => (
4   <Text key={index} style={globalStyles.bodyText}>
5     {paragraph}
6     {'\n'}
7   </Text>
8 ));
```

Additionally, an update to this backlog item was implemented in the following sprint. This update addressed some other leftover HTML code that was discovered later. In some articles, a piece of HTML code can be found which reads as \----- SPLIT ELEMENT -----. This was removed to increase readability for the users. The implementation is similar to the previous and simply checks if the next paragraph to be rendered is equal to the aforementioned remnant code, and if so it returns a null value and proceeds to render the next paragraph.

S1F12: As a user, I want to see breaking news with special styling

This PBI aims to implement the functionality of presenting breaking news articles with special styling. This feature is present on Ekstra Bladet's website but is absent in the design of the mobile application. Therefore, a design which was inspired by DR was created and accepted at a meeting by the product owner, Kristoffer. 13 Keep in mind that in the dataset provided by Ekstra Bladet's data science team, the articles do not have a property which indicate whether the article is classified as breaking or not. The product owner, Kristoffer, accepted this backlog item regardless at the aforementioned meeting.

S1B1: Construct a recommender model and train it on the EBNeRD dataset.

To get a headstart in development, and because the EBNeRD dataset was not available at the start of the sprint, a similar dataset was found, the MIND Mi-



Figure 13: An example of how breaking news are presented

crosoft News Recommendation Dataset (small), containing behaviour data for approximately 50.000 users and 50.000 articles. [32]

Other than resembling the EBNeRD dataset, the MIND dataset has numerous research papers documenting the AUC evaluation scores of numerous different machine-learning models. The top performing ones were LSTUR, NRMS, and DKN - all content-based filtering models. [63]

Therefore, a decision was made to attempt to implement one of these models. The implementation consisted of first training the models with the MIND dataset, evaluating the models, exporting the models, loading the models, and making predictions for a user in a realistic setting (how it would be done through an API call). Since all models were initially black boxes for the team, it was crucial to check if they could be used in the context of this project.

The group's main findings were the following:

1. (DKN, LSTUR, NRMS, etc. are specially made for MIND, the generated models are not compatible with PyTorch or Tensorflow, and the system requires major rework for other datasets.
2. These systems were not meant to make it into production. Going off the "intended path", and attempting to transfer any of the models into practical use is very painful.
3. Looking at academic papers, these systems give great results, but expanding our knowledge is also difficult due to insufficient documentation of the inner workings of the system and a lack of third-party resources.
4. Training and evaluating these models is incredibly slow.
5. Content-based filtering seems a lot less popular compared to collaborative

filtering, and therefore much more difficult to find helpful resources on.

Therefore, to be able to deliver value to the product owner with acceptable velocity, the group decided to look into collaborative-filtering-based models.

As stated above, collaborative-filtering-based models are the more popular and presumably easier option. This meant that there were more resources available for learning. Furthermore, not using one of Microsoft's models with rigid structures specifically made for the MIND dataset provided a lot more flexibility. Any issues during development were also easier to tackle because the tools are often common across different machine-learning systems. The team also got the impression that these models were also more scalable, and faster. Of course, a large trade-off is the quality of predictions, which is the most important aspect of the model - but being able to deliver a working product was deemed a higher priority.

As a starting point, the team used Daniel Pleus' (Machine Learning Engineer at Google) [14] PyTorch collaborative-filtering model [31] with the MIND dataset. As with the previous models, this first step was establishing that a full implementation was feasible. This meant that the model was trained and evaluated on the MIND dataset, could be exported, and loaded to make predictions for a single user. When this was successfully implemented, the data could be switched out for Ekstra Bladet's own.

The implementation includes loading the article data, history data, and behaviour data. Then binary labels are generated for clicks and no-clicks. Following that, users and articles are mapped with indices, which will be used as internal indices for the model. Both training and validation sets must share the same mappings to ensure consistency. Next, the data model and machine learning model itself are defined. As of now, only userIDs and articleIDs are used as embeddings - of

course, more features should be added later to improve recommendation quality. Next, the model is instantiated and trained.

S1B2: Evaluate the model using AUC and F1 scores.

F1 score and AUC scores are calculated using TorchMetrics modules. They are added to the model's code. A Tensorboard is also added to the notebook to log loss after each training and evaluation epoch. This is done to see if the model converges, or needs more training.

As expected, the AUC score is around 0.502, which is only slightly better than random guesses. Without any features, nothing better was expected.

After talking to the product owner, it was established that the only evaluation metric they are currently interested in is the AUC score.

S1B3: Make an API that returns a list of articles when requested for a prediction using the model.

The currently implemented API solution uses FastAPI, a modern web framework first released in 2018 for building RESTful APIs in Python. [17] On the backend side, endpoints and pydantic validation models defining how requests and responses should look are grouped in a single file, and utilities such as *make_predictions* and *get_all_news* are put into their utility folder. They are imported and called in the API file with the endpoints and pydantics. The trained model is imported to make predictions, and all articles and a single user are passed to the user for recommendation. In the future, the articles passed should also be more specific, as a user interested in sports is most likely not interested in sports news from several years ago. As of this sprint, a prediction request returns the 10 highest-rated articles for the user.

Axios is used to communicate with the API on the frontend side. [9] Axios is a library mainly used to send asynchronous HTTP requests to REST endpoints. The code for it is encapsulated in a file called `AxiosRequest` and is imported on screens where requests are made.

S1B4: Make the number of recommended news to a variable in the API call.

Changing the number of recommended news to a variable simply meant changing the request models, adjusting the Axios request accordingly, and passing the number in the request to the prediction function.

Appendix g

This section contains details about the rest of the completed PBIs from Sprint 2, which are:

- S2F1: Change the font in the article screen to one that looks better and is more readable (Sans-Serif).
- S2B1: Include meaningful features in the machine learning model to improve its performance.

S2F1: Change the font in the article screen to one that looks better and is more readable (Sans-Serif).

This PBI is a continuation of a previous backlog item regarding the implementation of fonts, which was addressed in the previous sprint. [2] As previously mentioned, there was a dependency on Ekstra Bladet's side, as the fonts for the screen showing the article could not be implemented before a design for that screen was made.

During this sprint, a design was made and shared with additional information

regarding which fonts were used in the design. It was then implemented using the same methods as the similar PBI from the previous sprint. A Sans-Serif font was implemented as this font family is generally considered more reader-friendly in body text.

S2B1: Include meaningful features in the machine learning model to improve its performance.

Early during the sprint, the team explored the integration of sentiment analysis as an additional feature within a binary classification model. The primary objective was to assess whether incorporating sentiment scores alongside user and item embeddings could enhance the model's ability to predict user engagement with recommended articles. However, after initial experimentation, and following advice from the supervisor, the team decided to pivot from this approach, opting for a more straightforward solution using an off-the-shelf model. The deciding reason for this was that the time saved on constructing a model and integrating more meaningful features would be better spent working on providing a better overall system. After all, the point of this project is not to build a machine-learning model from the ground, but to build a system that supports one. Furthermore, being able to construct this fully original model would most likely not give as good results as off-the-shelf options developed by a larger team over a longer period.

Appendix h

Data Preparation

In the following paragraphs, a step-by-step walkthrough is given describing how the data is prepared.

Load history and behaviour data This step is mostly self-explanatory. The EBN-eRD history and behaviour parquet files are loaded into pandas dataframes.

Make temporary dataframe containing all article_ids_inview for every unique user, because they are scattered in the behaviour dataframe The history dataframe contains a unique row for each userID with a column containing all off the clicked articles. What is missing is a row of all non-clicked articles.

Therefore, a temporary dataframe is made containing all articleIDs in view for each unique user. In short, what the code does is it groups by user_id, flattens article_ids_inview lists, and removes duplicates.

Listing 4: temporary inview dataframe

```
1 train_inview_temp = train_behaviour.groupby('user_id')['article_ids_inview'].agg(  
    lambda x: list(set(y for sublist in x for y in sublist))).reset_index()
```

Below is a more thorough explanation of the code. Throughout each of these steps, numerous sanity checks are done to ensure correctness.

`train_behaviour.groupby('user_id')`: This part of the code is grouping the dataframe `train_behaviour` by the column `'user_id'`. This means that it's grouping rows together where the `'user_id'` column has the same value.

`'article_ids_inview'`

: This specifies the column on which you want to perform aggregation. In this case, it's selecting the column `'article_ids_inview'` from the grouped dataframe.

`.agg(lambda x: list(set(y for sublist in x for y in sublist)))`: This part applies an aggregation function to each group. The aggregation function is a lambda

function, which is an anonymous function defined using the lambda keyword.

lambda x: This defines a function that takes a single parameter x, which represents the values in the 'article_ids_inview' column for each group.

list(set(y for sublist in x for y in sublist))): Within the lambda function, this code is creating a list of unique values by flattening the nested lists in each group's 'article_ids_inview' column. It iterates over each sublist in x (which represents the values in the 'article_ids_inview' column for a group), and for each y in each sublist, it adds y to a set to ensure uniqueness. Then, it converts the set back to a list to maintain the order.

.reset_index(): Finally, reset_index() is used to reset the index of the resulting dataframe. By default, when you perform a groupby operation in pandas, the grouped column(s) become the index of the resulting dataframe. This function resets the index so that the 'user_id' column becomes a regular column again.

Append article_ids_inview to history dataframe now that they are grouped by user_id The temporary dataframe containing all inview articles for each unique user can now be merged with the history dataframe on userID.

Make new column for unclicked articles by comparing clicked and inview articles in history dataframe Now that the history dataframe contains both all inview articles and clicked articles for every unique user, it is possible to extract non-clicked. This is done by adding every element to the new column from inview articles entry which are not in clicked article entry.

Listing 5: new column with all inview articles

```
1 train_history_with_inview['non_clicked_articles'] = train_history_with_inview.apply  
    (lambda row: get_non_clicked_articles(row['article_ids_inview'], row['
```

```
article_id_fixed']], axis=1)
```

Remove unnecessary columns A new dataframe is made only containing userID, clicked articles, and non-clicked articles.

Restructure data, so every interaction (click and no click) has a single row, and generate ratings column based on click or no click (1 for click, 0 for no click) Right now, each unique userID has only a single row containing all of their clicked and non-clicked articles. The next step is to create a new row for every interaction. This is done by iterating through every clicked article and non-clicked article for each user, and appending a new row to a new dataframe, which contains userID, articleID, and either 1 for click or 0 for noclick in the new rating column.

Listing 6: new row for every interaction

```
1 data = []
2
3 # Iterate over each row in merged_df_filtered
4 for index, row in train_history_with_unclicked.iterrows():
5     # For each item in article_id_fixed, add a row with rating 1
6     for item in row['article_id_fixed']:
7         data.append([row['user_id'], item, 1])
8
9     # For each item in non_clicked_articles, add a row with rating 0
10    for item in row['non_clicked_articles']:
11        data.append([row['user_id'], item, 0])
12
13 # Create the final dataframe from the collected data
14 train_history_with_rating = pd.DataFrame(data, columns=['userID', 'itemID', 'rating
    '])
```

Add topics column from article data using article_id (there are multiple topics for a single article) Next, using the articles dataframe, each row is appended with a new column containing all topics for a given articleID.

Listing 7: dataset

```
1 valid_history_with_topics = pd.merge(valid_history_with_rating, news[['article_id',
    'topics']], left_on='itemID', right_on='article_id', how='left')
```

Only keep the first topic for each article, and remove rows where topic is empty (and rename the column genre for now) To simplify the data, only the first topic is kept from the list of topics belonging to an article, and the few articles without topics are dropped. **If there are any duplicate entries for any userID itemID combination, remove the duplicates** This step is similar to many of the other sanity checks done throughout data preparation. If by any chance, any duplicate entries are detected - duplicates meaning userID articleID combination - they are removed from the dataset. **(Only for validation) Because the data is faulty, some rows are both present in training and validation datasets. We remove these from the validation data.**

This operation is done by running the following code in the notebook:

Listing 8: removing common rows

```
1 # Assuming you have imported pandas as pd
2 common_rows = pd.merge(valid_history_with_topics, train_history_with_topics, on=['
    userID', 'itemID'], how='inner')
3
4 if not common_rows.empty:
5     print("There are common rows between the two dataframes.")
6     print(common_rows)
7 else:
8     print("There are no common rows between the two dataframes.")
9
10 common_indices = common_rows.index
11 # Remove common rows based on userID and itemID combination
12 valid_history_without_matches = valid_history_with_topics[
```

```

13     ~valid_history_with_topics.set_index(['userID', 'itemID']).index.isin(
        common_rows.set_index(['userID', 'itemID']).index)
14 ]
15
16 # Reset the index of valid_history_without_matches
17 valid_history_without_matches.reset_index(drop=True, inplace=True)

```

In short, common rows between the two datasets are identified, and using index flagging, they are removed from the validation dataset. Below is a more thorough walkthrough of the process.

Merging dataframes: The code starts by merging two dataframes, namely `valid_history_with_topics` and `train_history_with_topics` on the columns `'userID'` and `'itemID'`. It performs an inner join, meaning it retains only the rows where there is a match in both dataframes based on these columns. The result is stored in the dataframe `common_rows`.

Checking for common rows: The code checks if the dataframe `common_rows` is empty or not. If it's not empty, it means there are common rows between the two dataframes, and it prints out those common rows.

Extracting common indices: The code extracts the indices of the common rows from the `common_rows` dataframe. This is done to know what rows should be removed from the validation dataset.

Removing common rows from validation dataset: Common rows are removed from the `valid_history_with_topics` dataframe. This is done by first setting the index of both dataframes, `valid_history_with_topics` and `common_rows`, to a combination of `'userID'` and `'itemID'`, then using boolean indexing to exclude the rows that are present in `common_rows`. The result is stored in `valid_history_without_matches`.

Resetting Index: Lastly, indices of `valid_history_without_matches` are reset to ensure a continuous index starting from 0.

This step is optional, because it is also done in before model evaluation as well:

Listing 9: removing common rows alternative solution

```
1 test_interactions_excl_train = test_interactions - train_interactions.multiply(  
    test_interactions)
```

Save data as CSV Both training and validation data is saved as comma-separated values files.

Now that data preparation is done, it is ready to be used with the LightFM model.

Firstly, training and validation data is loaded using pandas.

LightFM requires internal mapping of `userID` and `itemID`. The mapping is shared across training and validation to ensure consistency.

Listing 10: dataset mapping

```
1 dataset = Dataset()  
2 dataset.fit(  
3     users=pd.concat([train_data['userID'], test_data['userID']]),  
4  
5     items=pd.concat([train_data['itemID'], test_data['itemID']]))
```

Next, the model is instantiated and trained on the training set.

Listing 11: model instantiation and training

```
1 model = LightFM(loss='warp', no_components=NO_COMPONENTS,  
2     learning_rate=LEARNING_RATE,
```

```
3 random_state=np.random.RandomState(SEED))
4
5 model.fit(interactions=train_interactions, epochs=NO_EPOCHS);
```

The parameters are kept as they were in the Microsoft deep dive example, but should certainly be experimented with in the future.

S3F1: As a user, I want to see how much of an article I have previously read.

This minor UI improvement was implemented because it was a straightforward task given that the scroll percentage of an article is already tracked in the app. Although it does not directly contribute to the overarching sprint goal, it does give value to the product owner, further improving potential user satisfaction.

The design of the progress bar outside an article (on the feed) was primarily inspired by YouTube and Twitch, where the user's progress having previously watched a video is tracked.

Inside an article, this feature is essentially identical to DR Nyheder's progress bar, and it is also where the idea originated from. Users tend to check the length of an article before reading it, but by getting a feel of progression with a live progress bar, the user arguably has a more streamlined experience.

During a discussion with the product owner, it was also theorised, that if a user sees a half-read article, they would be more prompted to tap on the article again to finish it, having gained a sense of completion. Although this idea could be connected to psychological principles in gamification, it is most realistic to state that competitors' implementation confirms this feature's relevancy.

S3B4: Provide a visual representation for the training and validation AUC scores for each epoch to depict improvements and convergence.

Currently, matplotlib is used to plot training and validation AUC developments. To be able to track several models and save the data in a cleaner way, it would be a good idea to implement a TensorBoard instead.

Appendix i

S4B5: Improve performance monitoring by using TensorBoard.

As mentioned in sprint 3, up until this point, matplotlib was used to visualize the performance of the model. [9](#) And while this is a good general-purpose plotting library [\[30\]](#), it was decided to switch to TensorBoard as it is a tool designed with machine learning in mind.

TensorBoard can provide much more interactive visualizations such as scalar plots, histograms, distribution plots, and more. In addition, TensorBoard makes it easy to log the plotted data and compare the performance of multiple different models at once. TensorBoard was easily integrated into the current notebooks of this sprint, as the AUC scores per epoch were simply logged to TensorBoard and then visualized afterwards in the TensorBoard interface.

Appendix j

Finding a publicly available dataset to train and validate a potential recommender model on

Currently, it's uncertain whether the team will be able to obtain access to Ekstra Bladet's article and user behaviour data in time. Therefore, it's necessary to locate a comparable publicly available dataset. This will ensure the feasibility of training and validating a model using data that closely resembles Ekstra Bladet's if access is not granted.

The best candidate for this purpose is MIND: Microsoft News Dataset. The MIND dataset is available in demo, small, medium, and large sizes. The small size contains 150.000 rows of behaviour entries and 50.000 rows of new article data. MIND has been used in numerous scientific papers since its release in 2020, and seems to have enough data to support content-based filtering, collaborative filtering, and hybrid approaches. Furthermore, several implementations of all three kinds of recommender systems can be found on GitHub, and some of these are documented in scientific papers. [7]

Some other potential candidate datasets are the Globo, [33] Adressa, [1] and Plista [26]. Globo seems to be more fitting for contextual filtering models, and

Adressa and Plista do not seem to be publicly available, so they are not explored any further.

Appendix k

Context filtering

Alongside content-based and collaborative filtering, context filtering emerges as a promising method for enhancing recommendation systems.

Context filtering adds contextual information about the users' interactions to the recommendation process. By applying all of these criteria and contexts into a sequence, the recommendation system can better predict the probability of the next action.

In an example given by Netflix at the NVIDIA GPU Technology Conference, they trained a model on a sequence for a variety of users and their country, device, date and time they watched a specific film. From this, they were able to predict what the user would watch next. [37]

Although context filtering may not show as impressive results in itself as other options, a hybrid approach including contextual information could be a promising prospect.

Appendix l

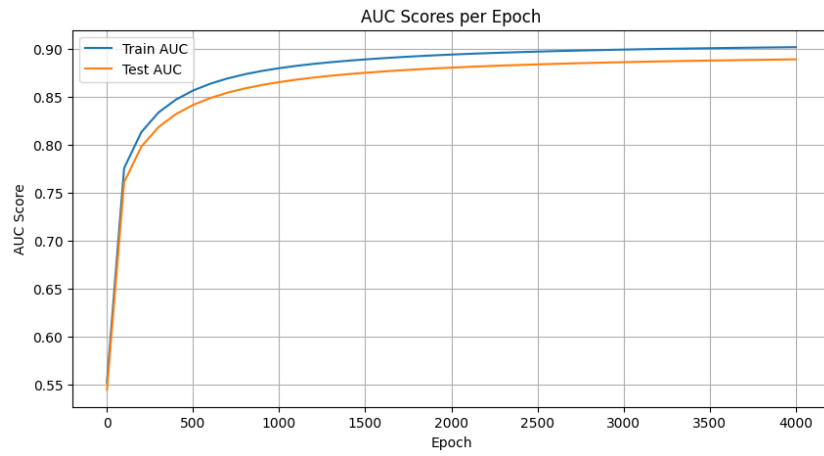


Figure 14: Model trained for 4000 epochs with 10 latent factors

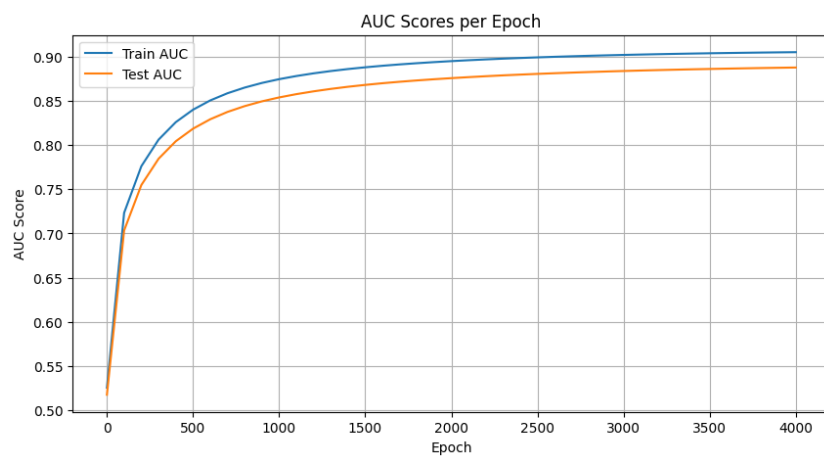


Figure 15: Model trained for 4000 epochs with 20 latent factors