UNIVERSITY OF TWENTE.

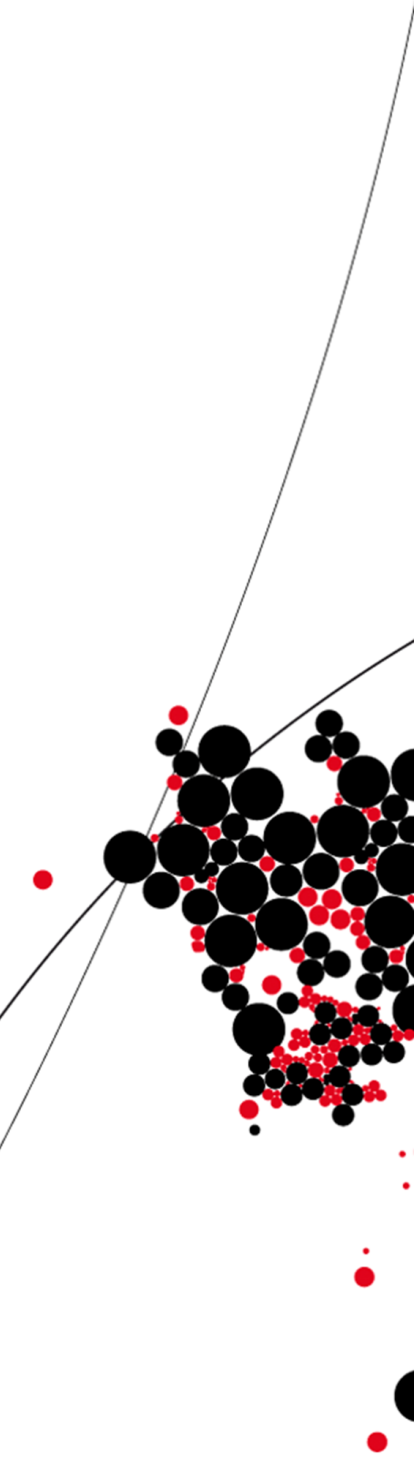
Faculty of Electrical Engineering, Mathematics & Computer Science

Understanding athletes preferences of sport activities through ranking based news feed recommendation

Stefano Perenzoni

Final project report

2021/2022



Supervisors:
dr. ir. D. Reidsma
dr. E. Mocanu

Human Media Interaction Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Abstract

Product personalization is essential to provide users with personalized content and retain them. Recommender systems are algorithms which give suggestions on what the users may find more interesting. In this thesis, we want to implement a recommender system to personalize adidas Runtastic's news feed. Firstly, we analyse the historical data of Runtastic apps to understand the business context and needs.

This thesis then proposes an approach to optimize diversity within ranking algorithms which does not rely on a diversification-aware loss function. Instead, we suggest Div-NDCG, a diversity-aware evaluation metric to be used in the model selection process to optimize both NDCG and the diversity metric by selecting the best model and dataset. Experiments are conducted on Runtastic historical data, from which we built multiple different datasets. The datasets are built through the combinations of different feature engineering and labelling techniques. Both heuristic-based and machine-learning models are then trained on tested on the datasets. By comparing their performances, we show that the diversity-aware evaluation metric better helps choose a model and dataset that optimize both metrics.

Finally, we propose an architecture to implement the tested models on the currently deployed Runtastic apps to provide product personalization.

Contents

Chapter 1

Problem Definition

1.1 Introduction

Runtastic is a digital health and fitness company, proprietary of the mobile sports apps adidas Running and adidas Training[1]. These provide the users with a tool to precisely track their activities and effort, while offering the opportunity to connect with other athletes and actively engage with them. The main goal of adidas Running is to provide a tracking tool for an endless list of sports such as running, cycling and team sports. Differently, the adidas Training app makes available a series of body workouts, with or without extra equipment. The apps also offer other sports and activation related features such as special challenges and monthly reports of the performed activities. Moreover, the two apps also share a social section, shown in Figure 1.1. Users can connect with their friends, have a look at the activities they have carried out, interact with them and show their support. As said, user profiles are shared between adidas Running and Training. Users indeed only require a single profile and the social interactions, as of added friends and posted activities, are synchronised through a news feed section shared between the two apps.

Currently, the news feed section of the adidas Runtastic's apps shows the sports activities performed by a user's friends in chronological order, according to the time they were completed, as shown in Figure 1.1. This is not optimal in the scenario in which we want to provide the users with the content they would most likely prefer, when he or she is scrolling through the social feed. For this reason, this research aims to analyse the feasibility of a personalised news feed. The goal of the final report is to show the relevance of a personalized news feed for Runtastic, explore and consider valuable alternatives for a recommender algorithm, implement and compare them based both on their offline performance and on the right fit they represent for Runtastic' business case.

[1] runtastic.com

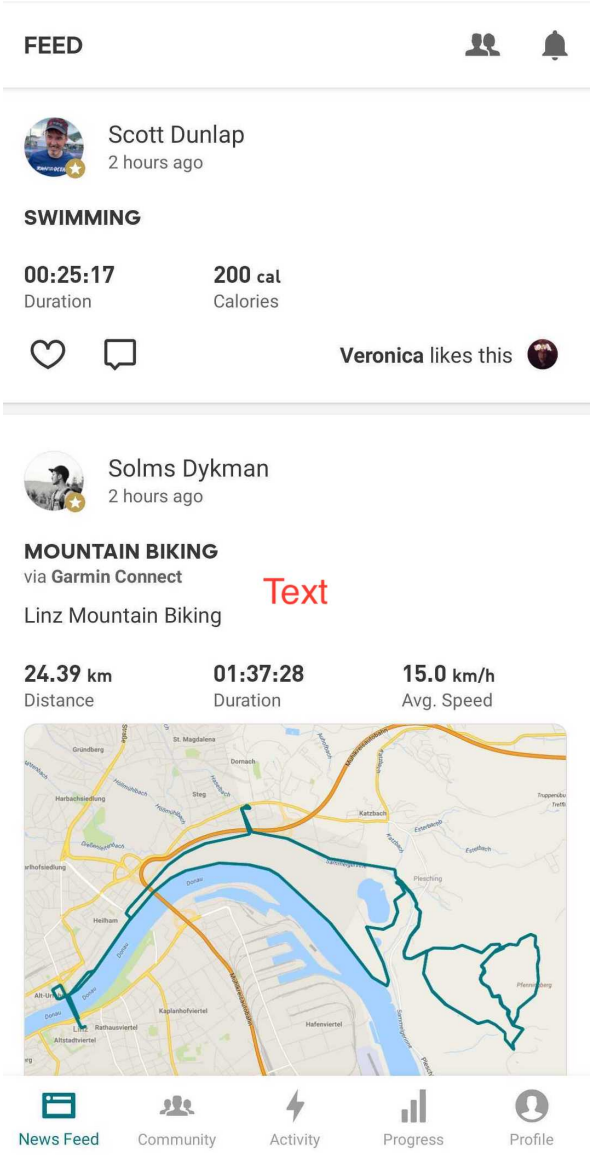


Figure 1.1: A view of the adidas Running news feed section.

1.2 Context and Background

After being acquired by adidas, Runtastic went through a rebranding process for both apps in order to better help adidas reach their purpose and goals. Even if the two, adidas and Runtastic firstly remained independent of each other, they work together with the goal of increasing adidas' brand credibility through better sport experiences and a community connected with the brand that makes the users feel connected between each other.

Creating a community of athletes is essential for adidas Runtastic for three main reasons:

- Community as strategy: In 2018, adidas launched a membership program,

which strongly moved the focus toward member experience. Runtastic wants to make its members feel part of the brand and its community. To do that, adidas offers personalized experiences to their customers by rewarding both sports apps engagement and purchasing activity.

- Community as retention tool: Runtastic strongly relies on the concept of community in order to activate its users, in the sense of doing sports and tracking their activities. Adidas uses adidas runner, physical communities of runners, to connect runners around the world. Complementarily, Runtastic has groups in its apps to form communities and let them interact digitally. Every group has the opportunity to define its own events in order to organize runs and activities. Moreover, adidas Running offers multiple challenges where a user can participate, track, and monitor his activities as an individual or as a group. Both challenges and groups have been proving essential to retain users and make them come back to the apps.

- Community as word of mouth: Online communities have been proven essentials for word of mouth communication [1]. Adidas also relies on the creation of communities to spread knowledge of its products. For example, adidas Running allows attaching ad-hoc pictures for each tracked activity. For each carried out activity, a user can attach up to ten pictures, showing, for example, the place where he went hiking. Finally, performed activities can be shared with a broader group of friends through more popular social networks, as shown in Figure 1.2

All these three components of community creation come together in the social features of the Running and Training app.

While Runtastic sports apps include many different features, users mainly use the apps because they offer them the opportunity to track their activities, check their progress and monitor how they are performing over time. Whether they are committed athletes or more sporadic ones, users might be seeking motivation from different components. For example, the adidas membership program aims at motivating sporadic athletes through the connection to the adidas brand. On the other hand, monthly advanced reports are offered by Runtastic in order to motivate committed athletes.

Additionally, being part of a bigger community is also a strong motivation. Users are motivated by being part of a community to which they feel connected. Groups and challenges, among the other features, work towards motivating users. However, reciprocal motivation mainly happens through interaction between users. Historical data suggests that users receiving interactions from their friends engage more with

Figure 1.2: An example of an automatically generated recap image which can be shared with friends on social networks.

the app, as shown in Chapter 3. Also, users giving out reactions to other people tend to later be more engaged and activate more.

So, with the goal of retaining more and more users, Runtastic wants to strongly focus on the social features in the two apps. The news feed section has been identified as a crucial step in the user journey in order to engage and constantly activate the users. Indeed, once a user spends time in setting up a social profile, completing it with a profile picture and a personal description, he or she expects to be able to connect with other users, see their relevant content and interact with them. Therefore, it is important to make a user's action visible to those who are following him and are interested in his sport activities.

User research was also carried out, by the user research team, on Runtastic users in order to understand their social mindset, both inside and outside the two apps. These include both surveys sent to a bigger audience and also one on one interviews to understand individual needs. Insights revealed that users strongly rely on the community of people they are connected to on the apps in order to find fitness motivation and inspiration. In a first survey about social profiles, out of about six thousand users of either app, 57.38% said that they use the social profiles in order to inspire others. 40.23% of the survey population also revealed that receiving interactions and comments from their social friends motivate them to go for the next activity.

The News Feed section of the apps, therefore, is the place where such motivation has to be found. It allows people to see friends' content and let them know whether they are impressed by a particularly impressive performance or by great

looking pictures attached to the news feed post. Facilitating a high number of interactions is essential for both the giver and the receiver of the same in order to improve motivation and engagement.

1.3 Problem

With the news feed being a major tool for engagement, we have to make sure that the users actually engage with what is shown them. The news feed is indeed the place where the content of social connections is consumed. In Runtastic's particular case, this happens through likes and comments. However, historical data has shown that only a few users actively engage with their community through social interactions, as explained in Chapter 3. Indeed, Runtastic measured a Social Interaction Rate (SIR) lower than ten percent in the last few months. SIR is a performance indicator that calculated the ratio of monthly active users who have at least given or received one social interaction.

Runtastic community-oriented strategy is strongly impacted by such lack of interactions. Indeed, without giving and receiving interactions, the sense of connected community does not fully spread amongst the users. Consequences are a weaker connection with both the other members and the brand, lower engagement, and a slower creation of online communities which results in a low level of word of mouth engagement of Runtastic products and initiatives.

Moreover, the lack of incoming reactions despite having multiple followers potentially represents a loss of motivation for social users, which also causes a decrease in social awareness. Indeed, previous studies suggested that receiving social feedback encourages users to increase their interactions towards other users [2].

1.4 Solution and goals

We believe that personalizing the News Feed by displaying to each user the most relevant activities for him can increase the impact of the apps and improve the products. Indeed, historical data has shown that users tend to interact more with particular kinds of activities, which are more relevant to some kinds of users than others, and therefore drive more interactions. For example, some users prefer activities where one or more images are attached, while others would rather see posts from their closest friends.

Recommender systems have changed the way users access, find, and interact with information in many fields. Users are now used to being targeted with tailored information meeting exactly their specific needs [3]. These models are learnt and

developed based on historical data to predict user interests with the best recommendation possible. To do that, recommender systems predict whether a user prefers an item rather than another in order to maximize the probability of the user interacting with the proposed item. Nowadays, they represent a win-win use case for both companies and customers [4]. The former are able to engage users more, increasing their value and minimize manual overhead. The latter are offered an improved service that is able to provide them what they really need rather than showing noisy and irrelevant information.

Recommender systems have been adopted in many different use cases, impacting almost every aspect of people's digital lives. Personalized content is showed users whenever they browse a search engine, or scroll through the products of their favourite e-commerce website. Also, smart recommendations became crucial within social network [3]. Here, these are involved in two use cases:

- Following or friendship recommendation, where a user is suggested a list of other people he might be interested in connecting with.

- News feed post personalization, where a user is displayed the content he is most interested in anytime he or she opens the app.

The latter is particularly important for most users who do not expect to receive content ordered chronologically, but instead sorted according to their interests. Indeed, the most obvious solution in case any kind of advanced algorithms is developed is to show the news feed in chronological order, showing at the top the most recent posts [5].

This is also the case for adidas Runtastic. Recommendation Systems could then be used to understand users' preferences and get the most relevant activities for them. The goal is to increase awareness and interactions in the news feed by showing the user the content they are more interested in. This thesis aims at studying how the technology of recommender systems can be applied to Runtastic's use case.

There are a number of technologies that allow us to personalize Runtastic's news feed: collaborative filtering methods based on the social interaction graph or content-based solution working upon a user profile and the characteristic of specific activities, for example. We need to understand what better satisfies our requirements.

The proposed solution should be able to increase the interaction within the News Feed by showing relevant activities at the top. However, there are two more constraints that have to be considered. Firstly, we previously mentioned the importance of receiving interactions in order to motivate and engage the users. We not only want to increase the social interaction rate, but we also want to maximize the number of different users receiving interactions. Diversity, formally defined later in Chapter 4,

is a particularly important metric. It is of interest of this thesis to include a measure of diversity within the evaluation of the proposed models.

So, we want to propose a personalized news feed with two main purposes:

1. Displaying personalized content to facilitate social interactions.

2. Displaying diverse content to ensure all users receive interactions.

In the second place, we want to propose a solution that can be put into production according to Runtastic capabilities and used in Running and Training apps to improve the products. Indeed, a fully personalised news feed would require a real-time use case. However, because of a longer data velocity in Runtastic's data sources, this is not trivial. Data velocity is the speed in which data is tracked on the apps, cleaned and made available, and this is approximately eight hours for Runtastic's apps. A possible solution can be scheduling the operations of feature engineering in order to have the most updated data available at run-time, for example.

In order to decide the requirements, we should consider the role of the news feed section within Runtastic' apps and want we want to accomplish. As said, activities are now shown chronologically with no personalization at all. So, a less complex rule-based model would most likely be already able to improve the amount of social engagement. At the same time, it would allow for a light and fast-to-run solution while providing great understanding of why some activities are ranked higher than some others.

However, researchers have shown how more complex machine learning approaches can push the performances even further, especially when trained over historical data describing the past user's interactions [6]. Chapter 2 deeps dive into an up-to-date literature study while giving particular importance to aspects relevant for our case. In particular, it discusses how recommender systems and ranking algorithms have been applied to personalize the content of a social news feed. Such investigation is necessary in order to fully address the main research question of this thesis.

1.5 Research questions

The contribution of this report is mainly practical and lies in researching how the Runtastic's apps can be improved. A second major contribution consists of researching how diversity within the recommendations can be optimized when selecting the best model.

Q1: How can the social news feed be personalized?

After the state of the art is understood, we want to develop a system that is able to rank the activities according to their relevancy to the user. With *activity*, we refer to any sport activity tracked by a user. Automatically, every activity is then displayed as a post in the news feed of those users following the one who carried out the activity itself. This problem can be formalized as following:

Given an input $x = (q, d)$ where q, called query, represents a user and d, called document, represents one activity, it wants to be identified a ranked list of multiple documents such that these are sorted according to a relevance score $s = f(x)$.

In this research, we want to explore two different approaches in order to compute such a relevance score: a rule-based model that relies on a series of heuristics and a machine learning based model trained over historical interaction data.

A heuristic based model can be defined through a set of manually inferred rules. Such rules can be inferred thanks to domain knowledge of both the data and the business case and also through interactive visualization techniques allowing us to identify relevant attributes related to a final relevance score [7]. Such a model can be used to clearly explain and motivate to the stakeholders the choices made by the algorithm. Moreover, the recommendations are more predisposed to any customization in order to diversify the proposed content.

On the other hand, as Chapter 2 will describe in more details, learning-to-rank machine learning models have been proved to be able to outperform rule-based ones. Learning-to-rank can be used to provide ranking-based recommendation of activities. The activities are then sorted in descendent order and displayed to the user starting with the activity the user is most likely to interact with. This approach allows us to leverage historical click data in order to infer which posts the users are most interested about.

Q2: How can the evaluated models be evaluated?

A first step for the evaluation of the proposed approaches is offline evaluation. Offline evaluation is performed on a portion of the data that was not included in the training dataset. This is used to assess how a model is performing with regard to some predefined evaluation

When it comes to understanding what is the best approaches, different point of views need to be explored. To start with, offline evaluation is necessary to understand how the model would perform on new unseen data in terms of ranking capabilities. These can be assessed through ranking metrics such as Normalised Discounted Cumulative Gain (NDCG), or Mean Average Precision (MAP). However, additional diagnostic metrics can provide a bigger picture and a clearer understanding of how the model behaves. These can still be measured in an offline environment, but are not strictly related to how the items are ranked. Differently, they can

describe how the models would help reach certain goals and needs, which drive the business case itself.

Q3: How can we guarantee results diversification in the chosen models?

We mentioned the importance of showing content coming from many different users, within someone's news feed. This is defined as results diversification. In our case, for each query, this is measured as the number of unique activity owners in the N most relevant activities.

This thesis wants to study how we can improve diversity within the chosen models. It starts with a literature review by looking at what was previously proposed.

Q4: What architecture design better fits the specific needs and the available resources?

Finally, a production-ready architecture wants to be designed according to the low amount of resources available. The chosen architecture has an impact on both the data pipeline and the applied models. For example, if a model has to be run asynchronously multiple times for each user, extracting complicated features out of a user past behaviour is made more critical.

This thesis wants to explore how personalization can be accomplished through the discussed models, and what other uses there are besides a fully personalised news feed.

1.6 Outline

The remainder of this report is organized as follows. In Chapter 2 an extensive study of the current state of the art is provided. I will go through many different aspects in order to provide a figure as complete as possible. These include recommender system, ranking-based recommendation, rule-based ranking algorithms, learning-to-rank machine learning models, and the application of the latter to the specific case of news feed recommender. Then, Chapter 3 gives strong fundamentals to the project and shows, through detailed analysis, why this is relevant for adidas Runtastic. Following, Chapter 4 explains the methodology followed for such project. It starts by explaining what algorithms have been considered and implemented and which tools were used to do so. Section 4.2 gives more details about the available data and how the dataset was collected. Then, it is explained how we will evaluate the proposed models. Chapter 5 shares the details regarding the technical implementations and what experiments, through the combination of different algorithms

and data configuration, were carried out. They are evaluated and results are reported. Finally, Chapter 6 goes into details of the proposed architectures, describing how the proposed model can be brought into production in Runtastic's apps.

Chapter 2

Related Work

This chapter summaries the results of the conducted literature study, comparing the available results with the current needs of our use case. To start with, a description of recent developments regarding recommender systems is given. Then, the specific case of learning to rank, and ranking-based recommendations are expanded in details. Finally, practical applications to news feed recommendation are discussed.

2.1 Recommender Systems

Recommender Systems have been gaining popularity with the increase of available digital contents on services such as e-commerces, streaming platforms and social networks. There are different definitions describing recommendation systems in different terms.

Ricci et al. described them as a combination of three main aspects: users, items, and interactions [8].

- A user is the target of the recommendation. A user can be represented by an ID or by features describing him and his behaviour. The most obvious attributes include information such as gender and age.

- An item denotes what the system recommends to the users. For example, this could be a product in case of an ecommerce or a post in case of a social network. They can be represented by a single ID or by a feature vector describing the item's content and characteristics.

- An interaction happens between a user and an item. This measures how a user responded to getting recommended a specific item by the system. It is also referred to as *feedback*. Feedback can be measured with two different methods: explicitly or implicitly [9]. Explicit feedback is expressed by the user

through a call to action. This includes, for example, ratings and reviews. Differently, implicit feedback is deduced from the user' actions and behaviour. Examples are watched content and clicked links.

The final goal of a recommendation system is to recommend items to a user for which an interaction is likely to happen.

There are many taxonomies categorizing recommender systems according to their characteristics. One of the most globally used ones is that defined by Burke et al. which divides recommendation systems into three main categories [10]. These are: collaborative filtering, content-based, and knowledge-based recommender systems. The first two, described in Figure 2.1 represent the most applied solutions within production environments [11].

2.1.1 Collaborative Filtering based

Collaborative Filtering (CF) is a technique used for recommendation systems. It is able to make recommendations for a user based on the opinions on an item of other users with similar interests [12]. There are two types of well known collaborative filtering techniques: user-based and item-based [13].

The first one makes recommendation for a user based on items interacted by or relevant to similar users. Similarly, the latter looks for items that are similar to those users liked in the past. These approaches further divides into two more categories. Collaborative filtering techniques can be either memory-based or model-based [14]. The former calculates the similarity between uses or items through metrics such as Pearson similarity or cosine similarity [15], [16]. Such models have been widely used in production thanks to their easy, affordable, and intuitive implementation. Also, they allow for easy integration in order to handle new data and scenarios. On the other hand, these approaches come with a series of limitations. A first, obvious, one is a direct consequence of the similarity-dependency. Indeed, these are not able to handle sparse data because of the lower number of common items between users. Moreover, they require the similarity to be calculated between each pair. Therefore, these techniques do not scale well for extremely large datasets. Thanks to the advancements of machine learning algorithms, model-based collaborative filtering have been studied able to both achieve better performances and scale more efficiently. These model use historical data such as rating or binary labels to train and learn a machine learning model and then make the final predictions. Many machine learning models have been proposed for such a task, for example, clustering models [17] and Bayesian classifiers [9], [18]. Recently, deep learning architectures have been applied to collaborative task with improvements in performance and more accurate recommendations [19]. He et al. proposed a neural collaborative filtering

based which was using a multi layer perceptron in order to model the interactions between users and items [20].

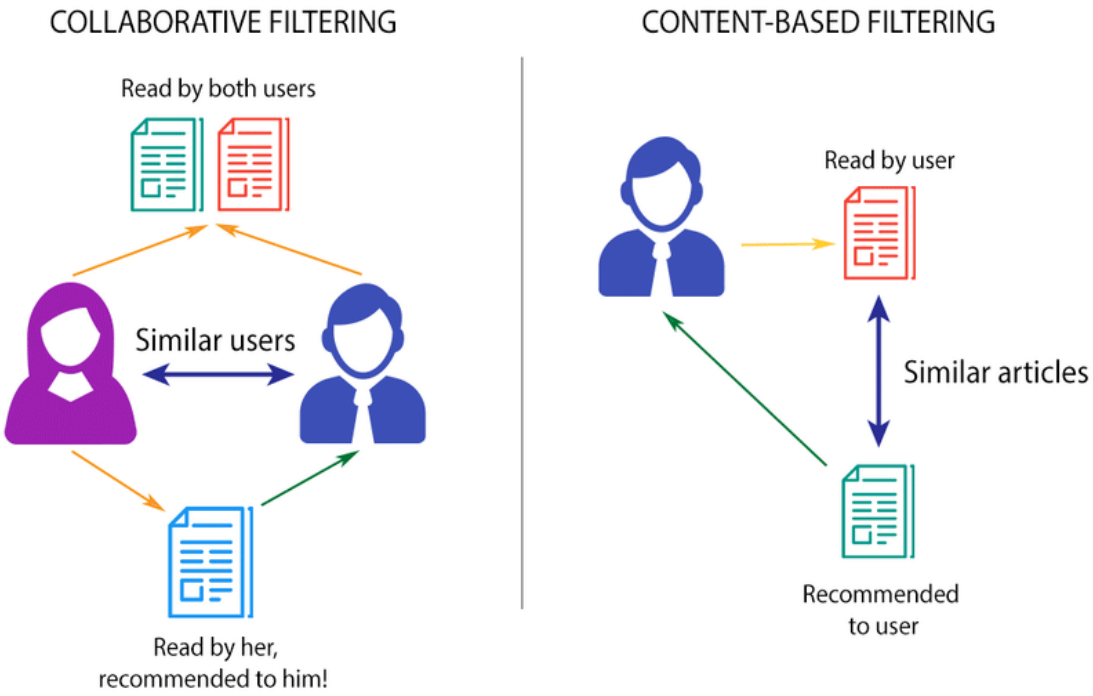


Figure 2.1: Recommendation logic for collaborative filtering and content based systems

2.1.2 Content based

Content based recommender systems make recommendations for a specific user by looking for items similar to those liked by the user in the past. The idea is to look at the aspects characterizing an item and identifying the main features that distinguish such item. These characteristics are used to build a user profile. Eventually, new items are compared with the user profile in order to decide what are the most similar items and recommend them [11]. Content based filtering techniques mainly work on two components: a user profile and an eligible item. Once the attributes of the two components are retrieved, the model can simply determine whether the item is relevant or not for the user [21]. For this final task, two approaches have been explored in the literature. Firstly, rule based models were used. These are defined through a set of heuristics normally used in traditional information retrieval methods [22]. On the other hand, recommendations are generated by a trained machine learning model. Here, a classification task can be planned if the target label is either interacted or non interacted. Differently, if the goal is to predict a continuos rating, regression algorithms can be used [23].

2.2 Learning to Rank

Traditionally, recommendation system were built using a rating score that was inferred by the user interactions. This was either binary or could have assumed multiple values. For example, in the popular MovieLens dataset, ratings ranged from 1 to 5 [24]. More recently, ranking-based algorithms were proposed and able to outperform rating-based ones in many use cases [25]. These models directly use ranking to recommend the most relevant items for users instead of ratings. So, ranking-based recommendation systems aim at ordering a list of eligible items according to their relevance. Differently, rating-based ones aim at predicting what is the rate a user would give to an unseen item and later sort such items based on the predicted rate in order to provide the final recommendations. In summary, the goal of learning-to-rank is to learn the best ordered list of items, while rating-based models aims at learning what is the exact rating a user would give to an item. While using the latter, a recommendation task can intuitively be cast as a ranking problem, just by ordering the items according to the predicted rating. In some applications, the exact rating may be not of interest, while more importance is given to the relative order of the items. As a solution, learning to rank (LTR) algorithms have started being used to build recommendation systems.

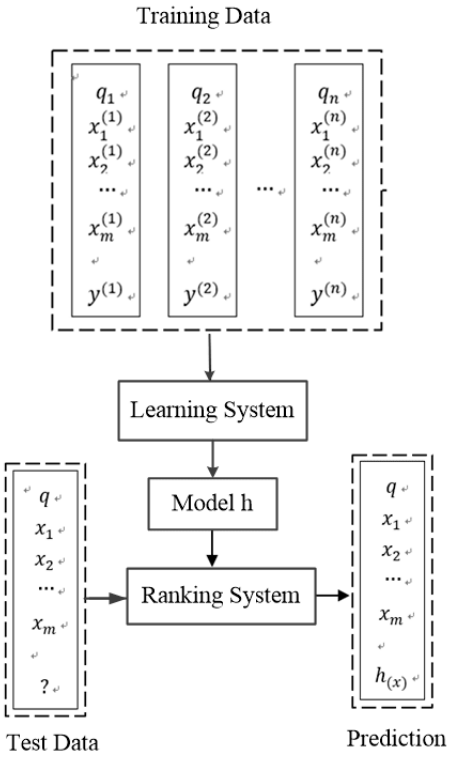


Figure 2.2: Typical supervised framework of a learning to rank algorithm [26]

This type of algorithms was not new within the literature, but it had already been

intensively explored within standard information retrieval tasks, such as search engine [27]. These algorithms are a branch of supervised machine learning. Normally, they require training data labelled with a relevance score, which defines the final order of the recommendations. Figure 2.2 shows the typical framework. The model is fed with queries q_i and each query is associated with many documents x_i. For each query, the ground truth y_i is represented by the optimally ordered list of documents. Ranking algorithms can be categorized into three main classes: pointwise, pairwise, and listwise. All the three methods demonstrated different characteristics in terms of performances and shortcomings [28].

2.2.1 Pointwise

Pointwise algorithms are able to learn to rank items from a particular mapping of the scores assigned by users to individual items, as a typical regression problem [28]. This is the first families of LTR that has been researched. These models do not learn the relative preference of a user towards items, but instead learn the absolute likelihood of a user's interests.

This approach uses a loss function typical of regression tasks, measuring the distance between the predicted relevance score $f(X_i)$ and the actual score y_i. For example, Mean Square Error (MSE) is used by many implementations of such algorithm [29].

$$MSE(S, Y) = \sum_{i}^{N} (s_i - y_i)^2 \tag{2.1}$$

So, loss terms are computed for each pair query-document and then summed together to compute the total loss.

The final ordered list is retrieved by sorting the documents by their relevance score with respect to the query Q. In the past, machine learning models such as Support Vector Machine (SVM) were applied to this problem [30]. Also, tree algorithms for order-wise classification and regression were also adapted [31].

Because of the foundation this approach is based on, there is clearly a gap between the actual objective of a ranking task and the training goal of pointwise ranking. These models are still able to reach good performances when applied to particular use cases. If the true relevance score for every query-document pair is well-known, then pointwise approaches can be a feasible solution. However, often such scores are not known but what is known is what items, those chosen by the user in the past, are more relevant than others, those ignored by the user.

2.2.2 Pairwise

Pairwise approaches formalize the ranking task as a classification problem by predicting which one of two documents is more relevant for a query. Therefore, the concept of absolute relevance score is replaced by that of relative preference. A pairwise model is trained by executing the following steps:

1. Takes document pairs correlated to the same query.

2. For each pair, a relative relevance label is calculated by observing the train relevance scores.

3. Trains a classification model from the labelled data

Eventually, the trained classification model is used to rank new unseen items. Intuitively, well known classification algorithms have been used [32]. Therefore, the learning objective of such models is that of minimising a loss function typical of classification problems, such as Binary Cross Entropy (BCE).

$$BCE(S,Y) = \sum_{i,j}^{N} y_{i,j} log(s_{i,j}) + (1 - s_{i,j}) log(1 - y_{i,j}) \tag{2.2}$$

Some known implementations of such technique are RankBoost and RankNet. The former operates in round, just like a AdaBoost algorithm. Over each round, a weak learner is trained. Finally, the weak learners are used to update the distribution between a pair of documents. Such distribution finally determines which item is ranked above the other one [33], [34]. RankNet, differently, is a deep learning approach which aims at minimising a probabilistic cost function by applying gradient descent methodologies [35].

On top of RankNet, better performing models have been explored and implemented. Burges et al. proposed LambdaRank, an extension of RankNet which only requires the gradients of the costs and not the costs themselves [36].

Besides outperforming pointwise methodologies, these models also shown some shortcomings. Firstly, the extremely high computational cost of generating the training samples limited the applicability of these models in real-world scenarios. A second limitation arises when such models are applied to data characterized by three or more relevance score. Indeed, because of the mapping to a classification problem, such algorithms and not able to catch and differentiate about the distance in relevancy between two documents.

2.2.3 Listwise

Pointwise and pairwise techniques are able to solve ranking tasks by mapping them to either a regression or classification one and minimizing a chosen loss function for these kinds of tasks. Therefore, the loss function does not match with the typical evaluation measures of a learning to rank task such as Mean Average Precision (MAP) or Normalized Discounted Cumulative Gain (NDCG), introduced in Section 1.1. Differently, the goal of listwise methods is to maximise the chosen ranking evaluation metric.

In order to accomplish that, many approaches were proposed. Taylor et al. proposed SoftRank. So far, we talked about deterministic relevance scores for each query-document pair. The idea of SoftRank is to predict a smoothened probabilistic score [37]. Such smoothening is used to calculate probability distributions for the final ranks of a document. Therefore, the training objective is an approximation of NDCG, called SoftNDCG, induced by smoothed scores. Differently, the ListNet algorithm uses Neural Networks and gradient descent to optimize the listswise loss function [38].

2.2.4 Diversity-aware Learning to Rank

Compared to graph-based solutions, learning-to-rank approaches are not as good as generalizing results diversification. Li et al. defined result diversification as the generation of a ranked list of items covering a broad range of topics, where a topic is a category describing the item, such as the genre in a movie recommendation use case [39]. Consequently, we refer to diversity as a metric which measures how good the model is at diversifying the results. As later said in Chapter 4, we proposed a diversity metric which counts the number of unique users among the top N activities in the ordered list.

Result diversification is normally guaranteed by the “next document” technique. The idea is to sequentially choose documents one-by-one based on the ones previously selected out of the ranked list [40]. It has been shown that such a task can also be included in the learning process. Reinforcement Learning to Rank has been proved successful [41], [42]. For example, Slivkins et al. proposed Multi-Armed Bandit settings have in order to select the best-ranked list that also maximizes user diversity [43]. However, as multiple list permutations are evaluated, such an approach is impractical at a product scale.

Other proposed solutions that iteratively select relevant documents to differ from those previously chosen according to a document similarity function through supervised machine learning methods. The main contribution of these proposals

is proposing a loss function that also optimizes diversity according to either user-defined similarity functions or by learning document similarity automatically [40].

Yan et al. demonstrated the benefits of a diversification-aware listwise approach by improved performances when compared to both pointwise and pairwise techniques [40]. However, the proposal of a personalized loss function goes beyond the scope of our project. Indeed, a diversity-aware loss function would add complexity to both the design and the implementation of the model. Moreover, the mentioned proposals aim at optimizing diversity in a setting with a high number of topics. Both Li and Wasilewski's works used queries characterized respectively by twenty and twenty-eight topics [39], [44]. However, in our case, the number of topics is upper bounded by the number of users someone else follows.

Moreover, as later explained in Chapter 4, our goal is also understanding how the different possible datasets impact the performances and what are the best settings for the production use case. Therefore, we differ from the above by proposing a method to optimize both diversity and ranking performance when selecting both the model and the dataset by using a linear combination of the two measures as evaluation metric.

2.3 News Feed Recommendation

News Feed Recommendation is one of the most important practical applications of recommender systems and ranking algorithms, given the popularity of social media platform nowadays. Studies have shown that more than one user out of two struggles with keeping up with the number of new news feed posts whenever they open an app. Also, they complained about being displayed non-relevant content with respect to what they are really interested in [45]. Ranking news post by relevance aim at solving this problem.

Supervised ranking models have been applied to get such relevance order. Therefore, in order to be trained, these models require the data to be labelled with a relevance ground truth. Many approaches were proposed regarding how to model such relevance score. A popular approach is that of relying on implicit feedback and handling relevance as a binary value. So, for a user, a post can be either relevant if the user interacted with it or non-relevant, if the user did not interact with the post. [46]. However, mapping the relevance score to a binary target also comes with some shortcomings. Mainly, a binary target is obviously not able to model the difference types of interaction a user can have with the posts. Indeed, the majority of social media platforms allow the user to interact with a post in many ways. For example, both liking and commenting a post is implemented in basically all social network. To fill such gap, industrial news feed ranking systems include multiple metrics and

feedback in the computation of the supervised label as a linear combination of the different feedback [47]. Combining multiple feedback brings the big advantage of being able to relate the relevance score more closely to the business goal of a project. For this reason, such approach is often applied in production environments. A third different relies once again on multiple feedback. However, instead of combining it within a single label, the multiple relevance scores are used to train different models which are later ensembled together [6]. Such approach allows understanding better how users' interests affect each of the feedback. Moreover, it does not deny the possibility of combining together the singular predictions by weighting them accordingly to the business goal. However, on the other hand, this obviously require more models to be trained and consulted for every unseen data point. Therefore, scalability might come less, especially in scenarios with limited resources.

As discussed in Section 2.2, a ranking algorithm takes as input a query Q and a document D. In the case of news feed recommendation, Q consist of the user who is viewing the posts in his news feed while D consists of the documents themselves. Therefore, modelling the user and the post is crucial. Historically, in machine learning models, the two components are modelled through a handmade feature vector. Within news feed, many features have been explored. To start with, a user is normally represented by his past behaviour and activity. This can be done not only by looking at the historical of posts a user interacted with, but also by looking at other behaviours of the user on the platform. For example, Chen et al. proposed a personalized tweet recommender system that strongly relies on the past tweets posted by the user himself. Also, more static parameters such as demographic of the users were proved critical in real world recommendation systems [48]. The representation of a post is also important. Intuitively, this can be represented through its content, such as text and images [49]. However, in many news feed, such as Runtastic's, activities are displayed with a quite minimal amount of information. In adidas Running and Training app, for example, each post shows the type of sport done in the activity, its duration, and some limited information about the physical effort made by the athlete, such as burnt calories. Also, a very short caption and attached images are allowed. However, these features are not very spread among our user base and General Data Protection Regulation (GDPR) requirements do not let us access such information. Therefore, a representation of a post strongly relies also on the representation of the user who performed it [50]. Finally, features modelling the interactions between the two users and other relevant social features started to be included [49].

More recently, deep learning architectures allowed the use of more complex embedding to model the user and the and post. Such frameworks made it more efficient to include negative instances within the representations. Wu et al. recently proposed

a neural networks based architecture in order to both incorporate multiple feedbacks and accurately distil negative instances in addition to the positive ones [6]. Eventually, the user and news embeddings are used to multiple relevance scores, which are then used to create an ensemble model.

This step of using the query and the documents modellings to retrieve the final relevance scores in the last step of a news feed ranking approach. As previously said, this can be eventually done through heuristic based models or through model training. Rule based news feed recommendation systems are still widely used in real world applications because of their flexibility, scalability, and explainability [51]. Berkovsky, for example, proposed a personalization algorithm which calculated relevance score by applying a variation of the tie strength model, which gives particular importance to the social circle of the user [52], [53]. Differently, many supervised ranking algorithms listed before have also been applied to news feed recommendation tasks. For example, De Maio proposed a pairwise LTR approach that uses users' past behaviours and tweets content to predict a relevance score calculated as a linear combination of feedbacks such as whether the tweet is retweeted or replied [50].

Chapter 3

Background and motivation

This chapter describes the results of the analyses that were performed in order to understand the contribution to Runtastic products and the feasibility of a personalised news feed section. Within Runtastic, every project is indeed prioritized according to how this relates to current goals and strategies. So, analysing the current use of the news feed, wanted to show how known Key Performance Indicators (KPIs) and metrics would compare between users not socially active within the apps and users who also actively used the social sections.

Some of the important concepts that will often be referred in this and the following sections are:

- Monthly Active Users (MAU): This first metric measures the number of users who opened and navigated the apps at least once over the month of reference. This is also calculated over a year (Yearly Active Users) or over a single day (Daily Active Users).

- Runtastic Attribution Model (RAM) Engagement: Within Rutastic's teams, the RAM is a well established and extensively used segmentation model which was developed by Runtastic data team. In particular, it is an RFM segmentation model, a model that differentiates users by looking at three variables: Recency (R), Frequency (F), and Monetary amount (M) [54]. In Runtastic apps all the three variables are related to the tracking of activities. Recency is measured as the days from the user's last activity, frequency measures how often a user performs an activity, and the monetary aspect indicates how intense the performed activities were by using the number of burnt calories. These three values are linearly combined to calculate the RFM score of a user. The RFM score of a user defines how engaged he is. To make it more readable to stakeholder and non-technical people, groups of users are created according to their RFM score. For example, Figure 3.5 shows three engagement groups. If a user has an RFM higher than a threshold T, he or she is classified as

retained and engaged otherwise the user is classified as retained and not engaged. Finally, if the RFM score is particularly low, the user is classified as lost. Users who are not lost are also referred to as *retained users*.

This is an example of a reduced version of the RAM to simply distinguish between new, not engaged, and engaged users. However, once the RFM value is calculated, more thresholds and therefore engagement labels can be defined. So, when referring to engagement of a user, we indicate the label the RAM has attributed him or her. Given a group of users, for example users tracking cycling activities rather than walking ones, and observing in what segments they are placed, we can understand how engaged, on average, a certain group is and compare it with other users.

- Time Spent in News Feed (TSNF): This is an important KPI when it comes to assess the importance of our social features. It measures how much time, in minutes, users spend looking at the news feed section over a specified period of time. Normally, this and other KPIs are calculated on a monthly basis.

Although this sections tries to give as much context as possible to motivate a personalized news feed in details, it is important to notice that much of the analysed data I was provided with is not publicly available outside Runtastic employees. Therefore, in the following paragraphs, approximations, ranges, and percentage ratios are used whenever absolute numbers are considered too sensitive.

Runtastic has been tracking and storing usage data since the creation of its products. Since scraping over the whole dataset requires days, we decided to analyse the data from a time span of one month, spanning from January 1st, 2022 to January 31st, 2022. In this way, we were able to perform calculations in a reasonable amount of time of a few hours or days while, at the same time, including a significant number of users.

A first analysis explored the importance of likes and comments in the news feed and how these reflect on users' engagement. A quantity-quality approach was chosen. Firstly, the users were divided into groups and the amount of users in each group observed, to understand how many tend to perform a particular action on the apps. Following, the average users' engagement, through RAM, is studied. In particular, we created and studied two groups of users: user giving reactions and users receiving reactions. Then, correlation between performed social actions and users' engagement was observed, for every group..

3.1 User giving reactions

We started by analysing the correlations between giving out reactions, likes and comments, and the engagement of a user. To start with, we defined a quantitative funnel to divide the users within groups. Figure 3.1 shows the user funnel in details.

Starting from the left side, there were about 5 millions users with at least one app session. We defined these as MAU earlier. Out of these, only approximately 10% have at least one active connection. With active connection, we refer to another followed users who had at least one activity over the period of focus. Nearly 358 thousands of these users actually uses the social feed section by scrolling their connections' activities. Finally, 57% of the users who see someone else's post also react to one or more friends' posts through a comment or a like.

Even though the biggest gap happens between MAUs and users with at least one active following, it is worth to notice that, out of the users with social connections, only 37% of them actually engages with their friends. So, on average, only one user out of three likes or comments a post.

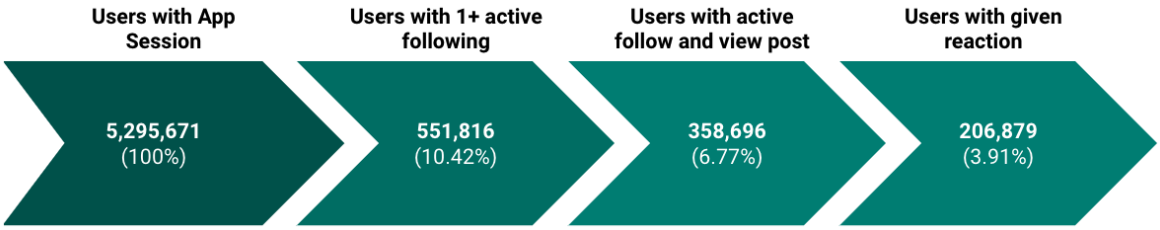


Figure 3.1: Funnel for users giving reactions

This first quantitative analysis shows us how the social features allowing users to interact with their friends are spread among Runtastic's users base. Differently, by qualitatively looking at the RAM segments of the different groups, we can notice correlation between being socially active and being engaged within the apps. Figure 3.2 shows, for each cohort, the portion of engaged users over the total number of users.

Besides the user cohorts introduced before, here we can notice two more. Firstly, users that view social posts but do not interact were considered. This cohort is obtained as a set difference between the set of users seeing friends' one or more posts and the set of users seeing and interacting with one or more posts. Doing so, we wanted to enlighten the difference between reacting and not reacting to activities. Then, on the furthest right, an additional group was considered for users at the 50th percentile regarding the number of given reactions. After observing an increase in engagement for users interacting, we also wanted to observe whether the users with a particularly high number of given interactions would result even more engaged.

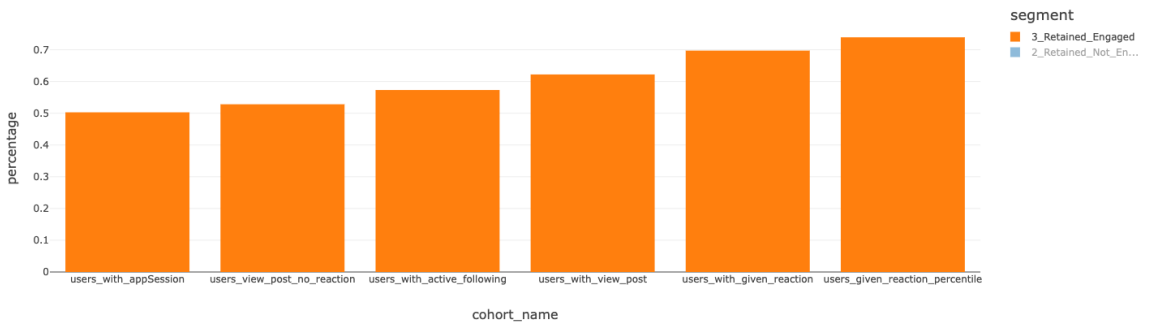


Figure 3.2: Engagement for users giving reactions

For the time span we chose, January 2022, this last group is composed of users that gave fifteen or more interactions.

Starting from the left, about 50% of users with at least one app session are engaged with the apps. This slightly increases for users with at least one active connection, remaining however below 60%. Also, users that use the social feed section and scroll through their connections' post tend to engage more. Next, there is a big gap in terms of engagement between users that react to the viewed post compared to those who do not react. Indeed, in the former group, seven users out of ten are engaged while for the latter, the portion of engaged users is, once again, about 50%. So, there is an improvement of about twenty percentage points between users who have given at least one reaction compared to users who see the news feed section but give no reactions.

3.2 User receiving reactions

A second analysis was performed focusing on analysing received reactions. The same approach as before was applied. Firstly, a user funnel was defined and evaluated quantitatively. Then, correlation between engagements and feature usage was observed. Moreover, further findings are presented regarding the time spent in news feed. For the previous analysis the TSNF KPI would have been biased as the way it is defined would make it circular when analysing users giving out reactions. Differently, for this one, we kept the KPI into consideration and looked into the average and median time users spend in the news feed per month, for each cohort. Table 3.1 summarizes these results.

Once again, Figure 3.3 quantitatively describes the users' funnel starting with MAUs, used as point of reference. About 70% of these users also tracked at least one activity over the period of time in focus. However, only 6% received a reaction by one of their connections. Only again, we can notice a big gap between users

with at least one activity, therefore users eligible to receive reactions, and users who actually receive at least one. Also, it must be noticed that the portion of users with a received view who also receive a reaction is quite big, about 75%. However, these users only receive sporadic interactions. Just about 20% of their activities are reacted.

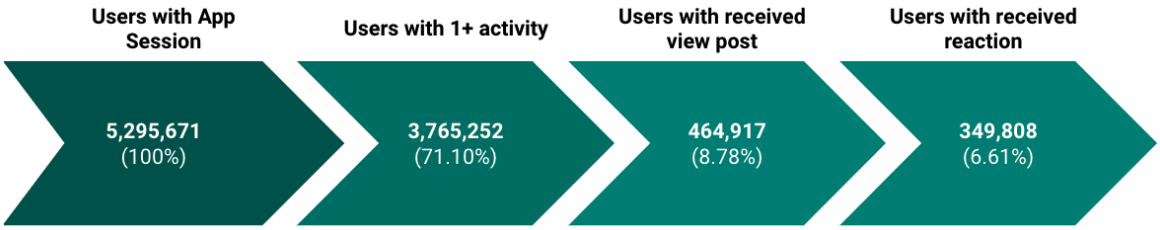


Figure 3.3: Funnel for users receiving reactions

Such behaviour also strongly impact the engagement of the users. Figure 3.4 shows the portion of engaged users for the different cohorts. The first bar regards users with at least one app session. There is immediately a big improvement in engagement while looking at users with at least one activity; about 70% of them are engaged. A most interesting gap is that between the third bar, users receiving a view but no reaction, and the fourth one, users receiving a reaction. Indeed, users from the latter group engage thirteen percentage points more than the other group. This shows us that there is actually a correlation between receiving reactions from your friends and being more engaged within the apps. Such correlation lies in the increase of the percentage of engaged users with respect to the reference cohort, which is represented by users with one or more app sessions.

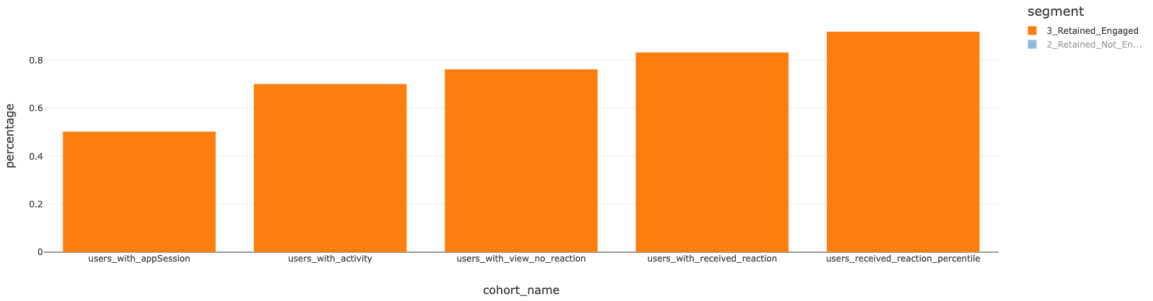


Figure 3.4: Engagement for users receiving reactions

As mentioned previously, correlation with time spent in the news feed is also relevant. This is indeed one of the main social KPI of adidas Running and Training as it gives a first measure of how important the news feed is for the users. Table 3.1 shows the different user cohorts and both the average and the median time they spend in the news feed, in minutes, ordered by the former. However, because of

Cohort name	Average TSNF*	Median TSNF*
users with app session	3.78	1.08
users with received view post	3.92	1.95
users with activity	4.11	1.15
users with received reaction	13	4.62
users with received reaction percentile	22.92	9.77

Table 3.1: Average and median minutes spent in the news feed section by cohort

a few outliers users who showed uncommon behaviours, averages could often be misleading. Because of that, we will look at the median. This starts at 1.08 for users with at least one app session. For the first three groups, no major changes are shown. Indeed, users with at least one activity also spend a similar amount of time in the news feed, showing no significant difference. However, there is a significant improvement between this group of users and those who receive a reaction. The amount of minutes users with at least one received reaction spend scrolling their news feed is indeed four times bigger than active users.

3.3 News Feed Retention

Finally, we carried out one more analysis to observe how general social features impact users' retention. Some general social features observed are, for example, adding a friend or commenting an activity. Each social feature defined one specific cohort of users. For example, social feature S_1 defines the cohort *Users performing* S_1. We used once again the reduced version of the RAM to observe how many users would retain and engage. However, this time we did not only looked at the portion of engaged users. Differently, we also wanted to observe how the amount of churned, or lost, users would change in each cohort.

To do that, we looked at new users and divided them into groups based on the app features they used over the first two weeks after registration. The analysis included about one hundred thousand new users registered between May 2021 and June 2021. For each feature related to social aspects, one cohort was created. These are displayed in Figure 3.5. As a user can performed multiple actions, these cohorts are overlapping. For each cohort, the portion of engaged users, in green, not engaged users, in orange, and churned users, in blue, is shown. On the left,

there are active users who did not use any social related feature over their first two weeks. About 60% of them later churned. It is worth to focus on users liking and commenting activities, respectively the third and the last bar. These two groups have approximately an engagement rate 45% higher than the baseline. Moreover, the portion of churned users is also relevant. Indeed, these are the two cohorts that shows the lowest number of lost users. In particular, only 6.84% of users commenting others' activities later churn. This is about ten times lower compared to the baseline. Therefore, this analysis once again shows a strong correlation between social features and retention.

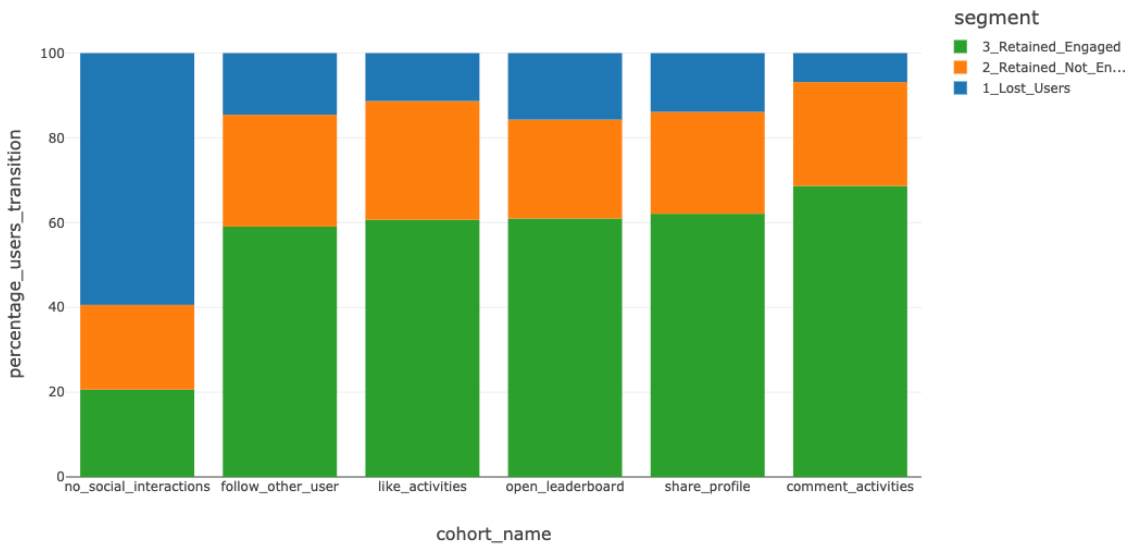


Figure 3.5: Runtastic Attribution Model segments for social cohorts

From the previously performed analyses, we are not able to point out specific causation between social features and users' engagement. However, we showed that users that are socially active, both in terms of giving and receiving reactions, tend to engage more and churn less. In the current situation, having a news feed which is not very used by Runtastic's users, the analyses' results are able to motivate the need of a personalised news feed section. Indeed, there are opportunities in order to improve both the amount of interacted posts and the amount of time spent in the news feed.

Secondly, higher engagement and higher time spent within the app are shown for users receiving one or more social interactions compared to those who do not receive any. Therefore, it is not only important to maximize the number of posts someone interacts with, but also the number of users who receive at least one reaction from one of their friends. This can be accomplished with a model that is able to diversify the content a user sees and interacts with.

To conclude, it is important to specify that all these analyses show correlation between the specific social feature and a metric of interests; users engagement, churn ratio and time spent in the news feed. However, as of now, we are not able to prove causality between such features and the improvements in the metrics. The only way we could prove causation would be by A/B testing over such social features. To do that, ideally, a group of users should be targeted with push notifications advertising the social section and their friends' activities. Then, the same metrics observed before should be studied for such group in order to identify how their engagement and retention changed after using the social section. However, A/B tests will not be performed in the work done for this thesis, but will eventually be done later in order to experiment over the models' results.

Chapter 4

Methodology

This chapter describes what algorithms were applied to personalize the news feed and the possible different approaches explored. Moreover, the complex structure of the dataset is explained. Finally, the evaluation metrics considered for the different models are introduced.

4.1 Ranking algorithms

This section describes what approaches will be followed for such personalized news feed. It starts by explaining why we ruled out some of the previously applied algorithms discussed in Chapter 2. Then, it describes the general algorithms that are selected and further experimented in Chapter 5.

In this project, we decided to focus on a ranking-based news feed recommender. Our goal is to implement a content-based method able to rank the different activities a user can see with respect to the content and the elements they display. So, based on the characteristics of the user, the activity and the user performing the activity, we want to calculate a relevance score. These three elements are used to retrieve a feature vector used as a representation of our data, as explained in Section 4.2.

We then excluded collaborative filtering based methods, which are often considered the state of the art for recommender systems [13]. One of the main reasons lies in the extremely high data sparsity of our particular use case. Indeed, data sparsity is known as a crucial disadvantage of collaborative filtering techniques [55]. With data sparsity, in this case, we refer to the lack of interactions between a particular activity and a user. Contrarily to what happens in e-commerce settings, where multiple users interact with one same item, in our news feed, the majority of activities do not receive any reaction and the majority of the users do not react, as analysed in Chapter 3. Therefore, both the social connection graph and the social interaction graph are extremely sparse and not suitable for a collaborative filtering based

approach.

On the other hand, the content displayed to the users for each activity is very self explanatory. The users see a limited amount of information such as duration of the activity and kilo calories burnt. Therefore, we want to leverage such contextual information by applying content-based ranking.

Another reason why ranking is preferred over traditional recommendations is the size of the pool of activities we want to rank and recommend. In our case, the size of such pool is very limited. Indeed, on average, adidas Running and adidas Training users do not follow many other people. We can assume we want to optimize the product for engaged users, who open the apps frequently. Hence, every time a user opens the app, there might be tens or hundreds of activities that need to be ranked. We do not expect such number to be in the order of thousands. Therefore, we can avoid using collaborative filtering as a candidate generation tool and focussing on ranking all the available activities.

So, the idea is to implement a personalized news feed through ranking. The goal of ranking is that of sorting a list of documents D according to a relevance score with respect to a query Q. In this project, documents are represented by the *posts*, also referred to as *activities*, as every post represents one and only one sport activity. Each query Q is represented by the specific user who has seen the posts in his news feed, regardless if he interacted or not with them.

This thesis mainly proposes two methods to do that:

- **Heuristic based model:** Rule based models allow us to define a set of rules out of personal knowledge of the data. Each rule regards one or more features and are processed together and combined to retrieve a final relevance rank.

- **Learning-to-Rank:** Leaning-to-rank models leverage past historical data to learn user's preferences. It is a supervised learning algorithm which goal is retrieving a sorted list of activities.

4.1.1 Heuristic-based ranking

A rule-based ranking model uses a set of rules. In our case, we designed each rule as a simple function. A rule takes one or more column of the feature vector as input and generate a single numeric output. Such outputs, coming from different rules, are then summed together to generate the final rank. Finally, the activities are sorted in descending rank values to define the final ordered list of activities.

We wanted to keep the model simple by using a limited number of heuristics. So, we only defined three types of rules. These represent three crucial components that

a user directly sees or derives, when seeing one activity in the news feed. The rules are the following:

- **Activity recency (R_1):** This is defined as $1/M_i$ where M_i is the amount of minutes from publication for an activity i. So, it measures how many minutes passed between the moment the activity was performed to the moment the activity was seen by a user in his or her feed. We take the reciprocal in order to give a higher value to recent activities.
- **Past interactions (R_2):** This is defined as the ratio of unique interactions, between the owner of the activity and the user scrolling his feed, happened in the N days prior to the moment the user is seeing the activity. Section 4.2 better defined the behaviour of such in-the-past dynamic features. This is the ratio of activity interacted over the total amount of seen activities. It is calculated for every pair user-user.
- **Activity content (R_3):** This last rule keeps into consideration the visual content displayed for each activity in the news feed. This value is calculated as the sum of some binary features and some other continuos ones. In particular, boolean flags are used to indicate whether the activity contained, for example, the map of the route or if the user attached any picture. The continuos features measure the particular sport performance. For example, a higher value is given to activities with a particularly high amount of burnt calories.

Finally, all the numeric rules are merged into a single ranking score by summing them together. We calculate the final relevance score as a sum of the rules as follows:

$$Score = \sum_{k}^{|K|} w_k * R_k \tag{4.1}$$

where K is the number of rules and w_k is the weight given to each rule.

Such mathematical model obviously resembles a traditional linear regression model. Indeed, linear regression and historical values could be used to learn the best configuration of weights. However, we decided to go for the heuristic approach and manually defined different configurations which prioritize different aspects of the activities. Chapter 5 uses one weight configuration to run the experiments.

4.1.2 Learning to Rank

The principles of Learning To Rank (LTR) were already defined in Chapter 2. The idea is to leverage historical data to encode a relevance score for each user-activity

pair. This can be done with multiple approaches discussed in Section 4.2. Then, labelled data is used to learn ranking how relevant each activity is to for a user. It is of our interest to try different implementations for each ranking technique mentioned before: pointwise, pairwise, and listwise.

Pointwise Pointwise LTR simplifies the problem by handling it as either a classification or a regression one, depending on the relevance variable. It therefore allows us to try many different known algorithms for these two tasks. Once again, the problem is kept simpler by relying on intuitive models such as logistic regression, tree-based models, and gradient boosting ensemble models. When the problem is mapped to a regression task, the predicted value is used to order the list of activities accordingly. Differently, when the problem is handled as a classification problem, two approaches could be taken. On the one hand, the activities could be treated as either relevant or non-relevant. Then, all the ones classified as relevant could be ordered with respect to some other values, for example recency. On the other hand, the probability score of an activity to be ranked as relevant, the positive class, can be used as relevance score and used to sort the list. We decided to apply this second approach, which allows us to rank the activities more precisely. Whether classification or regression algorithms are used depends on the type of label. For binary label, binary classification algorithms can be applied. In particular, the algorithms experimented in Chapter 5 are: logistic regression, Support Vector Machine (SVM), random forest, and gradient boosted trees. Differently, when the relevance label can assume multiple values, both discrete or continuos, regression models are applied. The algorithms tried in this project are: linear regression, SVM, random forest, and gradient boosted tress.

Pairwise Pairwise ranking transforms the ranking tank into a classification one by considering pairs of items. Within this project, we apply LambdaMART, which is a boosted tree-based variation of LambdaRank [36]. This approach can be applied both to a binary relevance label and a continuos relevance score, allowing us to try different datasets and configuration. LambdaMART is an ensemble method of Multiple Additive Regression Trees (MART) which applied gradient boosting to learn how to order a pair of items at a time [36]. An XGBoost implementation of LambdMART was used for our experiments[1].

Listwise Through a listwise approach, we want directly optimize the selected ranking metric. We used the LambdaLoss framework in order to incorporate the metric

[1] XGBoost Ranking

objective into the ranking task [56]. Such framework let us leveraging LambdaRank algorithm for optimizing metric-driven loss functions which are directly related to the task we want to solve. So, by defining the specific loss functions, we are able to perform listwise ranking using metrics such as Normalized Discounted Cumulative Gain (NDCG), or Mean Average Precision (MAP), as objective. Section 4.3 explains how the metrics work and why they are relevant. The problem with these metrics is that they are rank-dependent and need to be sorted which makes the metrics non-differentiable and therefore not suitable as learning functions. LambdaLoss is a probabilistic framework that assume the scores of documents S determine a distribution over rankings. Given S, the probability of obtaining relevance labels Y and the probability of a ranked list π are modelled. This finally allows taking both ranks and scores when defining the loss function [56]. An XGBoost LambdaLoss implementation to directly optimize NDCG was used in this project.

Finally, we want to leverage distributed computing in order to train and tune the models and evaluate multiple hyperparameter configurations in parallel. Therefore, tools such as Spark, scikit-learn, and hyperopt are combined. In parallel parameter tuning, the same data is propagated to different nodes of the cluster. Each node can then train and evaluate a centralized model. The parameters optimization is done, for all models, by directly using an evaluation metric as objective function. Finally, the best configuration can are tested on an unseen test dataset.

4.2 Dataset Extraction

The dataset represents a big complication of this project because of two reasons. The amount of data available at adidas Runtastic is tremendously high. In details, data is tracked and collected for most of the users' interactions within the app. For example, for each sport activity, more than two hundred attributes are collected and persistently stored. This makes it impossible to compute the data locally and makes it necessary to use modern distributed computing framework such as PySpark [2].

So, both data collection and feature engineering take a consistent amount of time. To partially overcome such limitation, we opted to focus on a subset of the data in order to develop and evaluate a Proof of Concept (PoC) model. So, one month of user impressions and interactions is considered. Also, only users meeting certain criteria are involved. For example, we want active users with at least a minimum number of active connections. However, such filtering is applied at the end of our data pipeline so that we are able to retrieve data for all the users.

[2] Apache Spark

The data pipeline is built as sequential blocks. A block is dependent on the previous ones and therefore executed sequentially. It starts by retrieving the IDs of the user database. Once the first datasets of IDs has been defined, the features and the remaining data can be retrieved. Firstly, impressions are retrieved. An impression is the action a user performs when seeing an activity in its news feed. Each impression indeed represents one entry in our dataset. Once impressions are retrieved, we look for interactions, likes and comments. So, out of the past posts a user seen, we retrieved which ones he liked or commented. These are marked as two boolean columns in our impressions' dataset. So, to every interaction there is a correspondent impression. However, not all impressions also have an interaction. Those impressions with no associated interaction represent the negative instances of our dataset. We want to include both positive and negative instances, since it has been proved this approach normally improves performances and reliability of the model [57].

Once these components are defined, the pipeline proceeds with retrieving the relevant features to represent the pairs query-documents, or, user-activity. Here, we can identify three types of features characterizing each entry.

- **Query-related features:** These are strictly related to the user behaviour, as he represents the query. They are also referred to as *user context*. User context could either be static or dynamic. The former consists of those features that are independent of what happened in the past, such as country, gender, and age of the user. Differently, dynamic features are used to model how the user behaved in the past N days regarding, for example, the most interacted sport and the most done sport. N can be chosen dynamically.

- **Document-related features:** This set of features is related to the activity shown in the news feed post. They are also referred to as *activity context*. These contain anything relevant in order to describe the activity. Obviously, particularly important are those aspects actually shown on the post, such as calories spent or time of the activity. Then, they can also model whether an activity belongs to a day streak or whether it is a particularly special performance, for example.

- **Query-Document interaction features:** These features model everything related to the interaction between the query and the document. Mainly, they describe past interactions between the user viewing the post and the one who performed the activity. For example, how many interactions happened between the two users in the past N days. Again, N can be arbitrarily chose.

It is of our interest to explore different values of N. We expect that a higher value gives more importance to the past while decreasing the novelty, differently, a lower

number prioritizes the content of the post itself. In particular, Chapter 5 goes through two different values for all our datasets.

When supervised machine learning algorithms are used, our feature vector need to be matched with a label which represent the final score of the pair. This label is obtained as combination of the possible interactions a user can have with one activity in the news feed. In our case, a user can like or comment an activity. There are multiple possible choices on how to encode such target label. In particular, we identified three different techniques to incorporate together such information.

- **Bipartite ranking:** The type of interaction is ignored, and the label mapped as a binary feature. It is either 0, if the user did not interact with the post, or 1, if the user either liked or commented the post. Despite being the most intuitive way, this solution is not able to describe the difference between the two actions.

- **Cumulative ranking:** The relevance label is calculated as the sum of the types of interactions occurred. So, the label can assume three values: 0, if no interaction between the user and the post happened; 1 if the user either liked or commented the specific activity; 2 if the user both liked and commented the post. Such method allows us to prioritize activities that trigger multiple reactions by placing them at the top of ordered list.

- **Weighted ranking:** The relevance label is calculated similarly as before, as a linear combination of whether a like and a comment happened, with arbitrary weights. So, while previously the weights were equal to 1 for both types of interactions, this third method allows us to arbitrarily choose the weighting in order to prioritize different interactions. For example, Section 3 showed that comments minimize the probability of churning more than likes. So, scoring comments more than likes would prioritize the former, giving these activities a higher score.

In Chapter 5, three different datasets, each created with one of the previous methods, are used to train the selected models and later compared. In this project, we limited ourselves to just consider liking and commenting as possible interaction. However, our target label could be improved by considering other types of implicit interactions the user has with the post. For example, dwell time could be considered [58]. Dwell time is defined as the amount of time a user spends looking at a post in his news feed. While we initially considered including it, we finally decided to exclude dwell time after observing some data quality issues within our tracking data we would have used to calculate the metric.

4.3 Evaluation

The models, separately trained on all the obtained datasets, are then evaluated through offline evaluation. Offline evaluation is performed on a portion of the data that was not included in the training dataset. This is used to assess how a model is performing with regard to some predefined evaluation metrics.

In our case, we want to focus on two kind of metrics. Firstly, we want to assess how accurate the ranked list. So, metrics to measure the relevance of ordered items are used. In a second place, we want to compare the models on metrics more relevant to the business case. In particular, we are interested in measuring how diverse the made recommendations are.

4.3.1 Ranking performances

Mean Reciprocal Rank (MRR) This metric is one of the most typical ones when it comes to evaluating ranking models [59]. It is defined as the mean of all the users' reciprocal rank (RR). It evaluates the reciprocal of the rank at which the first relevant activity was retrieved [59]. It is equal to 1 if an activity was retrieved at rank 1, 0.5 if it is retrieved at rank 2 and so on. It is calculated as follows:

$$MRR = \frac{1}{|U|} \sum_{u}^{|U|} \sum_{d}^{|D|} \frac{1}{rank_d} \tag{4.2}$$

where U represents the set of users and $|U|$ its module and $rank_d$ is the reciprocal rank of document d for the user u. Despite being one of the most observed metric, it does not ideally fit our problem. Indeed, MRR efficiently measures performances of models where the user is only interested in one relevant document, the most relevant one. MRR does not consider the rest of the ranked list. Differently, our models returns a list of activities ranked by their relevance, for which we want the user to browse in its totality.

Mean Average Precision (MAP) This metric is normally used to evaluate binary relevance models. So, when the list of documents is ranked through a binary feature indicating either relevancy or non-relevancy to the user. The MAP is indeed calculated using concepts of binary classifications such as confusion matrix, and, in particular, precision. Precision measures the percentage of relevant activities in all the retrieved items.

$$precision = \frac{TP}{TP + FN} \tag{4.3}$$

where TP stands for True Positive and is the number of correctly relevant classified activities, while FN stands for False Negative and is the number of correctly activities classified as non-relevant.

Differently from MRR, MAP evaluates multiple items of the list up to a specific cut-off N. Such cut-off N is included into the metric with the precision at N (Precision@N) metric. This measure the fraction of relevant activities in the top N recommendations. Such value allows the user to personalize the evaluation metric, deciding how many elements we want to limit the evaluation to and can therefore be adapted accordingly to the specific case. For example, if the users of Runtastic's apps views about 10 posts per session, on average, we could set the cut-off to 10 to have a clearer view of how our model would perform in the real-world scenario. Mean Average Precision is therefore calculated as follows:

$$MAP = \frac{1}{|U|} \sum_{u}^{|U|} AP(u) \tag{4.4}$$

$$AP(u) = \sum_{k}^{N} precision(k) rel(k) \tag{4.5}$$

where n is the total number of activities in the ranked list, precision(k) represents the cut-off to precision@k, and rel(k) is set at 1 if the item at tank k is relevant, 0 otherwise.

As said, this metric handles binary ratings where, for a user, an item is either relevant or not relevant. However, in our news feed, we want to rank activities placing at the top those the user is most likely to interact with. Such interactions consist of either likes or comments, or a combination of the two. Therefore, mapping it as a binary classification problem would not allow capturing the full picture.

Normalized Discounted Cumulative Gain (NCDG) This last metric has a similar approach to MAP and give more importance to highly relevant activities that are ranked at the top of the list. NDCG is able to handle cases where items are ordered according to a relevance score, which is not binary any more. It evaluates the models keeping into consideration that highly relevant activities (e.g. activities that could trigger both a like and a comment), should precede medium-relevance documents (e.g. activities that would only trigger a like). These should obviously precede items with no relevancy at all (e.g. activities that would not trigger any interaction)

NDCG is calculated through the combination of the concepts of gain (G), cumulative gain (CG), discounted cumulative gain (DCG), and ideal discounted cumulative gain (iDCG).

Gain is defined in Formula 4.6 and simply measure the relevance score. In our case, it consists of the numerical ratings given by the combination of the explicit feedback given by the users through likes and comments.

$$CG(k) = \sum_{i}^{k} G(i) \tag{4.6}$$

Cumulative gain is calculated as the sum of singular gains up to the first k elements. k represents the length of the considered list of recommendations.

$$DCG(k) = \sum_{i}^{k} \frac{G(i)}{\log_2(i+1)} \tag{4.7}$$

Discounted cumulative gain makes up for the main short come of CG which does not consider ordering of the items. Differently, dividing the gain by the rank accounts for the position where an activity was placed in the list.

$$iDCG(k) = \sum_{i}^{|REL_k|} \frac{G(i)}{\log_2(i+1)} \tag{4.8}$$

where REL_k consists of the optimal list of relevant documents up to position k, ordered by relevance.

$$NDCG(k) = \frac{DCG(k)}{IDCG(k)} \tag{4.9}$$

Compared to the previous two metrics, NDCG is able to consider both the absolute position of an activity in the ranked list and its relevance, represented through a non-binary relevance score.

4.3.2 Recommendation performances

According to the business reason explored in Chapter 3, we want to evaluate our systems accordingly. The types of chosen metrics strictly depend on the particular goal we want to achieve. Therefore, we needed metrics able to evaluate the systems in areas not related to the users' historical interactions.

Diversity If we want to maximize the number of users receiving a reaction, the evaluation of diversity within the recommended users is extremely relevant. It can be

defined as the number of different recommended activity owners among the top K items. So, we want to measure the number of unique users such top recommended activities were performed from.

We calculate diversity as a percentage so that it can be represented as a real number between 0 and 1 without further scaling, just like the other aforementioned metrics. Diversity is defined as the number of unique users divided by the maximum between K, the cut-off used for evaluation, and the number of other users followed.

Other metrics could be of interest, for example, coverage could be used to measure what portion of the followed users gets recommended. We decided to just stick to diversity when it comes to business related metrics as it is one of the most important objective of our news feed section.

4.3.3 Evaluation Metric

We propose a linear combination of ranking metric, Normalized Discounted Cumulative Gain, and diversity as evaluation metric to optimize both measures while selecting the models' parameters and the dataset. Such metric, referred to as Div-NDCG, can be trivially calculated as follows:

$$DivNDCG = w_1 * NDCG@10 + w_2 * Diversity@10 \tag{4.10}$$

where 10 is the cut-off K chosen for the evaluation, while w_1 and w_2 respectively weight the ranking and diversity metric.

Chapter 5

Experiments and Results

This chapter describes the experiments and discusses the results obtained by the proposed models on the different datasets. The experimented models are: rule-based ones, scikit-learn and gradient boosting implementations of pointwise models, a gradient boosted tree implementation of the LambdaMark algorithm and a gradient boosted tree implementation of a listwise model using NDCG for optimization within the loss function through the LambdaLoss framework.

Different datasets were built out of one month of tracking data on both adidas Running and adidas Training apps. More details about the datasets are given in Section 5.1.

We evaluated each dataset on the metrics mentioned earlier, with a particular focus on NDCG and the diversity-aware metric with weights $w_1 = 0.7$ and $w_2 = 0.3$.

5.1 Dataset

We used the data pipeline described in Section 4.2 to build the experiment datasets. One month of tracking data was used, from January 1st 2022 to January 31st 2022. As adidas Running and adidas Training share a common social section, tracking data coming from either one or the other was treated in the same way. Each row of the dataset, called *impression*, consists of a user seeing one specific activity. An impression is referred to as *interaction* if the user interacted with the displayed activity. Despite the dataset type, the target label is zero for an impression and an integer bigger than zero for an interaction. These were calculated with the three methods described in Section 4.2. All the impressions of a user are identified by a unique pair $(user\_id, activity\_id)$. So, the single key $user\_id$ identifies a query, and it is composed by all the impressions given by that specific user.

Time Window. Flexible time windows were used to extract dynamic features dependent on past behaviour. We used two different settings: seven days window and thirty days window. The latter is a recurrent value in Runtastic's data products, which allows us to speed up the data pipeline. On the other hand, with the former, we wanted to try a different approach which gives more importance to recent users' behaviour. At the same time, because of a lower number of distinct user another one can have interacted with in seven days, we expected higher diversity with this setting.

Data preprocessing. We followed a preprocessing approach to reduce the size of dataset by keeping relevant types of users, integrate the dataset with features coming from other table of Runtastic's database, and encode the features. One-hot encoding was applied for categorical features and scaling for the numerical ones. Finally, we filtered out whole queries based on the number of total users followed and interactions given. Only users following more than five other users and who have given at least ten interactions were kept.

After applying the time window and data processing, two datasets were created. The datasets were of the same size, containing the same users and the same activities. They differed in how the features were collected. The first one used a time window of 7 days, while the second one a window of 30 days. Both had three different types of labels. Table 5.1 summarizes the characteristic of the main dataset, from which the others have been derived. Finally, the dataset was separated with an 80/20 ratio on the number of users into training and test datasets.

# Users	# Activities	Size	Median # activities per user
13224	2684254	$1.6GB$	101

Table 5.1: Main dataset size and properties

For what regards activities receiving any type of reaction and activities with no reaction, the dataset is balanced. Out of the whole dataset, 52% of activities received an interaction through either a like or a comment from the user of that specific query. Most of these interactions are likes, as comments are much rarer in Runtastic' apps. Indeed, amongst the activities with received interaction, only approximately 10% are comments. About 60% of the activities that triggered the user's comment also triggered the like reaction.

5.2 Experiment Setup

The experiments were run in a distributed computing environment in order to speed up the hyperparameter process. The used cluster was composed of 4 workers for a totality of 64 cores and 448 GB memory. For the ML models, model selection was parallelized across the cores using Hyperopt, an open source tool for automated parallel hyperparameter tuning [60]. As we used single-machine ML models, Hyperopt was used to run in parallel models with different parameters. Tree of Parsen Estimators (TPE) algorithm is a Bayesian method used for parameter search while optimizing the diversity-aware evaluation metric. TPE creates probabilistic models out of the history of previously evaluated hyperparameters to later suggest the next hyperparameters to evaluate [61]. Bayesian optimization is generally sequential, as the results of the previous runs have to be known to model the next run's parameters. So, this is parallelized by modelling the *parallelism* parameter. This parameter sets the maximum number of parallel runs. By having parallelism between 1 and the maximum number of configurations, we are able to parallelize the hyperparameter process while also exploring larger parameters ranges thanks to the TPE algorithm[1].

Both NDCG and Diversity were evaluated with the same cut-off $K = 10$. Training and hyperparameters optimization was done on the training dataset by using 5-fold cross validation and the best parameters chosen. Then, the models were evaluated on the same test dataset kept as unseen data. The python library ranx was used for the evaluation of the ranking metrics [62]. Ranx allows us to evaluate different models on the same dataset and compare them by performing statistical tests. It offers a series of in-built ranking metrics such as MRR or NDCG. Moreover, we implemented and integrated our define diversity metric.

We decided to keep a single hold-out test set for the final evaluation, and applied statistical significance tests to compare the different models and detect statistically significant improvements. We followed the approach recommended by Smucker et al. and applied randomization test through Fisher two sample randomization test [63], [64].

5.3 Bipartite Ranking

This first dataset has a binary label. This section shows how the selected models performed on both the 7 days and 30 days dataset. For the three ML-based approaches, the best performing model on the validation sets is evaluated on the test dataset. In the following tables, only the performance of the best performing algo-

[1] Scaling Hyperopt

rithm for each LTR approach is reported. The heuristic-based model uses equal weights for the three rules explained in Chapter 4.

Table 5.2 shows the performance of the selected models on the ranking metrics for the 7 days window's dataset. In the table, every approach tried is identified by a character. Then, the superscripts over cells' values indicate whether, for that specific metric, the approach of that particular row significantly outperformed the others according to the statistical significance test applied.

#	Model	MAP	MRR	NDCG@10	Diversity@10	Div-NDCG@10
a	Chronological baseline	0.613	0.647	0.530	0.720^{bcde}	0.587
b	Heuristic-based	0.668^{a}	0.682^{a}	0.595^{a}	0.645^{cde}	0.610^{a}
c	Pointwise ranking	0.759^{ab}	0.872^{ab}	0.784^{ab}	0.591	0.726^{ab}
d	Pairwise ranking	0.764^{abce}	0.883^{abce}	0.795^{abce}	0.604^{c}	0.738^{abce}
e	Listwise ranking	0.757^{ab}	0.879^{abc}	0.787^{abc}	0.611^{cd}	0.734^{abc}

Table 5.2: Results on MAP, MRR, NDCG@10, Diversity@10, and Div-NDCG@10 for the bipartite ranking dataset with a 7 days time window.

The chronological baseline is the currently implemented news feed, which shows activities ordered chronologically, starting from the most recent ones. While achieving high diversity, the baseline lacks in the other ranking metrics, indicating the shortcomings of such method in facilitating social interactions. The heuristic-based model is able to significantly increase the ranking metrics MAP, MRR and NDCG@10. However, it comes with a drop in Diversity. Given the weights used for the Div-NDCG metric, overall, the latter outperforms the former. The three machine learning models perform similarly, and they also outperform both the baseline and the heuristic one on the ranking metrics. In particular, the pairwise ranking model performs the best on all three metrics. Despite a lower diversity, compared to the listwise method, it performs better in the diversity-aware NDCG metric.

Table 5.3 summarizes the results for bipartite ranking on a dataset created using a 30 days time window for the dynamic features.

By collecting dynamic features over 30 days, the heuristic-based model achieves a Diversity@10 close to the one of the baseline, while obviously outperforming the baseline on the other ranking metrics. Once again, the pairwise method outperforms all the other models for what regards the three ranking metrics. However, it performs poorly on diversity and consequently in the diversity-aware NDCG. Because of that, despite a lower NDCG@10, the pointwise method outperforms the pairwise one on

#	Model	MAP	MRR	NDCG@10	Diversity@10	Div-NDCG@10
a	Chronological baseline	0.613	0.647	0.530	0.720^{bcde}	0.587
b	Heuristic-based	0.664^{a}	0.673^{a}	0.584^{a}	0.690^{cde}	0.616^{a}
c	Pointwise ranking	0.763^{abe}	0.886^{ab}	0.791^{ab}	0.613^{de}	0.738^{abde}
d	Pairwise ranking	0.773^{abce}	0.890^{abce}	0.802^{abce}	0.553	0.728^{abe}
e	Listwise ranking	0.758^{ab}	0.884^{ab}	0.789^{ab}	0.562^{d}	0.721^{ab}

Table 5.3: Results on MAP, MRR, NDCG@10, Diversity@10, and Div-NDCG@10 for the bipartite ranking dataset with a 30 days time window.

the Div-NDCG evaluation metric. This shows that including diversity within the optimization evaluation metric over the stages of parameters tuning and model selection help select a model able to achieve good diversity while only showing a slight drop in NDCG.

When comparing the performances over the two datasets, the models trained over the 30 days dataset result in improved ranking performances. In particular, the ranking model shows an improvement in MAP and NDCG@10. Pointwise ranking shows better performances in both NDCG@10 and Diversity@10. On the other hand, for the other models, the improvement in NDCG@10 is followed by a significant decrease in Diversity@10.

5.4 Cumulative Ranking

This second dataset has a label built as the sum of the types of possible interaction happened. Having only likes and comments, here the label can assume three different values. Both a 7 days time window dataset and a 30 days one were created. For the pointwise approach, we modelled the problem as a regression task rather than a multi-class classification.

Table 5.4 starts with the results for the tried models on the cumulative ranking dataset with a 7 days time window. Here, the same chronological baseline model performs with a decreased NDCG compared to bipartite ranking. This is a consequence of the truth relevance scores ranging amongst three values instead of two.

On the other hand, MAP and MRR, which only consider whether an activity is relevant or not and do not account for its relevance score, show results on the same scale as before. Also, while NDCG decreases, Diversity does not follow the same trend compared to the bipartite ranking results but instead it shows improvements

#	Model	MAP	MRR	NDCG@10	Diversity@10	Div-NDCG@10
a	Chronological baseline	0.613	0.647	0.460	0.720^{bcde}	0.538
b	Heuristic-based	0.661^{a}	0.682^{a}	0.514^{a}	0.636^{cde}	0.550^{a}
c	Pointwise ranking	0.763^{ab}	0.889^{abd}	0.721^{ab}	0.598	0.684^{ab}
d	Pairwise ranking	0.768^{abce}	0.897^{abce}	0.724^{abc}	0.613^{ce}	0.691^{abce}
e	Listwise ranking	0.763^{ab}	0.890^{ab}	0.723^{ab}	0.599^{abd}	0.686^{ab}

Table 5.4: Results on MAP, MRR, NDCG@10, Diversity@10, and Div-NDCG@10 for the cumulative ranking dataset with a 7 days time window.

for different models. Once again, the heuristic-based model shows improved performances on all ranking metrics and decreased Diversity@10 compared to the baseline. Consequently, the former only performs slightly better than the latter on the diversity-aware evaluation metric. The three ML-based techniques come with an improvement in raking metrics but a decline in diversity. However, the decline in Diversity@10 is small. In particular, compared to the heuristic-based solution, pairwise ranking is able to achieve a NDCG@10 of 0.724 with drop only to 0.613 in Diversity@10. The Pointwise and listwise models perform similarly to each other. Also, ranking metrics are similar to the pairwise one but they come with a lower Diversity@10. Compared to bipartite ranking, the ML-based models achieve better performances for both MAP and MRR suggesting an improvement in recommending relevant activities which tend to drive any type of interaction.

In Table 5.5 are shown the results of cumulative ranking on the 30 days window's dataset. For all ML models, the trend is similar to what was observed in bipartite ranking. There is an increase in ranking metrics compared to the shorter time window's dataset. In particular, the three ranking techniques show a significant improvement in NDCG@10. Once again, the pairwise model performs better than the others in both NDCG@10 and Diversity@10.

5.5 Weighted Ranking

This third dataset's label is obtained through the weighted combination of the different types of interaction. We weighted commenting an activity twice the weight of liking an activity. So, there are four possible values for the relevance label. Once again, regression was preferred over classification for the pointwise ranking approach.

Starting with table 5.6 and the results on the weighted ranking dataset, we can

#	Model	MAP	MRR	NDCG@10	Diversity@10	Div-NDCG@10
a	Chronological baseline	0.613	0.647	0.460	0.720^{bcde}	0.538
b	Heuristic-based	0.664^{a}	0.673^{a}	0.514^{a}	0.690^{cde}	0.567^{a}
c	Pointwise ranking	0.776^{ab}	0.897^{ab}	0.741^{abe}	0.595	0.697^{abe}
d	Pairwise ranking	0.780^{abce}	0.903^{abce}	0.743^{abe}	0.617^{ce}	0.705^{abce}
e	Listwise ranking	0.774^{ab}	0.895^{ab}	0.739^{ab}	0.607^{c}	0.700^{abc}

Table 5.5: Results on MAP, MRR, NDCG@10, Diversity@10, and Div-NDCG@10 for the cumulative ranking dataset with a 30 days time window.

#	Model	MAP	MRR	NDCG@10	Diversity@10	Div-NDCG@10
a	Chronological baseline	0.614	0.646	0.412	0.720^{bcde}	0.504
b	Heuristic-based	0.661^{a}	0.682^{a}	0.449^{a}	0.636^{cde}	0.505
c	Pointwise ranking	0.766^{abe}	0.891^{ab}	0.659^{abde}	0.603	0.642^{abd}
d	Pairwise ranking	0.766^{abe}	0.890^{ab}	0.644^{ab}	0.618^{c}	0.636^{ab}
e	Listwise ranking	0.759^{ab}	0.895^{abdc}	0.655^{abd}	0.625^{cd}	0.646^{abcd}

Table 5.6: Results on MAP, MRR, NDCG@10, Diversity@10, and Div-NDCG@10 for the weighted ranking dataset with a 7 days time window.

notice that NDCG@10 is even lower with respect to the results obtained in the previous two datasets. Because of that, the heuristic-based model does not outperform the chronological baseline according to the diversity-aware evaluation metric. Amongst the machine learning methods, pairwise ranking performs poorer than the other two on the ranking metrics, in particular on NDCG@10 where it is outperformed by both pointwise and listwise models. Also, the listwise approach performs well in terms of diversity. Thanks to its performance on Diversity@10, the listwise method has the best performance on Div-NDCG@10. Once again, by including diversity in the optimization metric, Diversity@10 gets prioritized and improves by allowing the selection of a model able to perform well in both NDCG and diversity. Compared to the results on the previous datasets, despite a general decrease in NDCG@10, we can observe an improvement in Diversity@10 for all the tested methods. Regarding the other ranking metrics MAP and MRR, the weighted ranking has similar results as the cumulative ranking, which, as observed earlier, had improved results compared to the binary one.

Finally, table 5.7 reports the results obtained for weighted ranking on the dataset with a monthly time window. An improvement in ranking metrics for the machine learning based solutions is also observed in this third dataset. Indeed, the models here outperform the models trained on the previous two datasets. The three learning-to-rank techniques all perform similarly on the ranking metrics. Pairwise once again achieves slightly better performances for what regards MAP and MRR. However, the listwise approach results in the highest Diversity@10 while also outperforming the other on Div-NDCG@10.

#	Model	MAP	MRR	NDCG@10	Diversity@10	Div-NDCG@10
a	Chronological baseline	0.614	0.646	0.412	0.720^{bcde}	0.504
b	Heuristic-based	0.664^{a}	0.673^{a}	0.461^{a}	0.690^{cde}	0.530^{a}
c	Pointwise ranking	0.774^{abe}	0.981^{abde}	0.675^{abd}	0.616	0.657^{ab}
d	Pairwise ranking	0.780^{abce}	0.901^{abe}	0.672^{ab}	0.617	0.655^{ab}
e	Listwise ranking	0.769^{ab}	0.894^{ab}	0.677^{abd}	0.627^{cd}	0.662^{abcd}

Table 5.7: Results on MAP, MRR, NDCG@10, Diversity@10, and Div-NDCG@10 for the weighted ranking dataset with a 30 days time window.

Chapter 6

Architecture

This chapter describes how the previously introduced models can be used within the adidas Running and Training products. Firstly, their possible applications are discussed in Section 6.1. This does not only focus on a fully personalized news feed, but also discusses other uses for the proposed ranking models. Then, Section 6.2 and Section 6.3 describes how the proposed use cases would be implemented, dividing them into two families: non-real-time and real-time applications. Finally, Section 6.3.1 describes how the process of feature engineering can be handled in order to be able to make real-time prediction within an environment with limited resources.

6.1 Personalization

The work of this thesis started from the idea of proposing a personalised news feed section that could have replaced the current one, based on chronological ordered activities. However, that is not the only way in which an activity ranker model could be used on the adidas Runtastic products to engage the users even more. Indeed, the final ordered list retrieved as output from the proposed models allows personalization at many different levels within the apps. For example, this could be used to improve, through personalization, already existent features. On the other hand, it can also be used to expand the product and offer new capabilities. Following, three different applications are described regarding how a ranked list of new friends' activities can be used to leverage tailored content.

- **CRM Notification** Customer Relationship Management (CRM) through push notifications represent a fundamental tool to activate users. However, they have to be used carefully as users can be easily bothered by not relevant content. The ranked list of non-seen activities can be seen to decide what is the most relevant one and target a user with personalized content, inviting him

or her to interact with the activity.

- **Relevant Activities Section** A new section can be added to the Runtastic apps showing the most relevant friends activities a user might have missed. Ideally, such a section would contain a very limited number of activities while keeping the news feed ordered chronologically, with no changes. This could also leverage diversity by, for example, only showing a maximum of one activity for each user.

- **Fully Personalized News Feed** The sorted list of ranked activities can be used to fully replace the chronologically-ordered News Feed by displaying first the most relevant activities. Therefore, every time a user updates the news feed, the non-seen activities are ranked and displayed accordingly.

While the first use case can work asynchronously, the last two work synchronously with the client side requests of the users. In detail, CRM push notifications send out time is not dependent on the usage of the apps by the user. The ranking model can be run at any time on a user and the most recent activities his or her friends has performed. Then, a notification can be sent to the user asking him to check out and interact with the highest ranked activity. Differently, for the other two use cases, all the new activities a user can see have to be ranked and ready to be displayed at the moment the user access the apps. These two different scenarios are defined respectively as asynchronous and synchronous recommendations. The main difference between the two lies in when the activities have to be ranked, which impacts the way all the necessary data is made available at runtime for the model to be run.

6.2 Asynchronous Recommendations

The concept of *asynchronous* recommendations lies in the fact that the recommendations are not a response to an action performed by a user in the app. Differently, the recommendation can be performed at any preferred time; asynchronously with the behaviour of the user in the app. So, asynchronous recommendations are not triggered by client requests coming from the user's device.

Given a user and a list of new activities performed by his or her friends, asynchronous recommendations can be made at any desired time by running a saved model. The concept of saved model is crucial for this section. With saved model, we refer to any model ready to be applied to get a ranked list of activities. For example, this can simply be a vector of weights or a trained machine learning model. The models discussed and evaluated in Chapter 5 can all be used as a saved model.

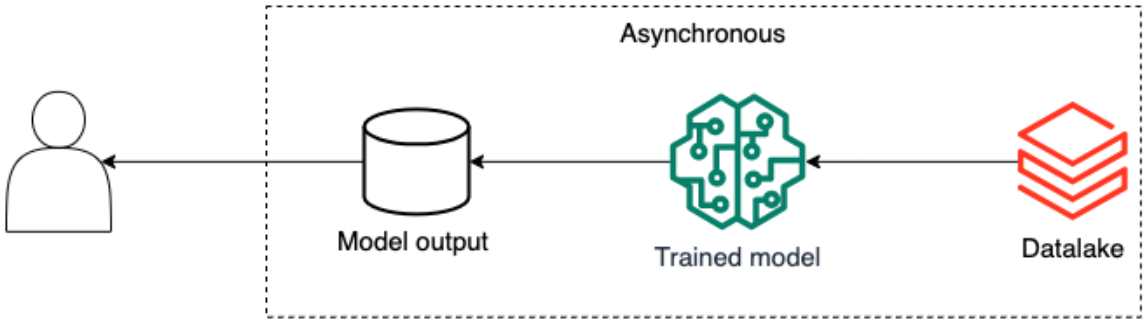


Figure 6.1: Flow diagram for asynchronous recommendations.

In an asynchronous use case, the advantage is that both the runtime of the model and the time necessary to retrieve the needed data are not relevant. The only requirement is that the two steps are executed one after the other, starting from the latter. As an example, let's have a list of N users U for which we want to rank a list of activities A_i with $i \leq N$. As said in Chapter 4, the ID of the user U_i and the ID of the activities in A_i uniquely identify a query-document pair and are all we need to retrieve the necessary data.

Once the data is retrieved, it can be given as input to the saved model. For each user U_i, the output is the list of activities A_i ordered by relevance. Finally, the most relevant activity can be sent to the user as the content of a push notification. In this way, the users are targeted with content which they might actually interact with. Figure 6.1 shows a simple schema of the described process.

6.3 Synchronous Recommendations

Synchronous recommendations are the response to a user's action performed on the apps. Whenever a user opens the social news feed or refreshes it, his friends' new activities have to be ranked and displayed to the user ordered by relevance. This happens through a continuos communication between the client, the backend servers, the recommender model, and the data sources. Each component has to perform rapidly.

The proposed architecture does not change client and backend sides significantly. The two components would follow the same structure as the current news feed. The only slight difference would require the client side to display the activities ordered according to their rank, which however does not represent a major issue.

On the other hand, both the saved model and the data sources have to be designed accordingly. This thesis does not analyse in details the runtime necessary to run each model. Differently, we assume the models discussed in Section 5 would

only add low latency to the current process. Our assumption is supported by relevant publications which showed a limited time complexity of the various models early discussed [65], [66], [67]. Moreover, Runtastic relies on Amazon Web Services (AWS) to effectively manage AI as a service for its products [68].

At last, compared to the asynchronous setup, the major difference lies in the data sources and in the way data is retrieved. We assumed fast response by the saved models once all the necessary input data is given. However, retrieving the data from Runtastic data sources is not trivial in time and its time complexity cannot be assumed to be irrelevant to the whole process.

In this section, we focus on how the data is made available in run time for our models. We start with the data pipeline described in Section 4. The section introduced three families of features: **document-related**, **query-related**, and **query-document interaction** features. Moreover, a further differentiation was made between static and dynamic features. As said, the latter are the result of the aggregation of multiple values over a time window.

It is important to understand what tables of Runtastic's relational data sources are involved. For query-related features, two tables are involved. One describes the user and all the information available about him or her, such as gender, age, or location. This is where all the static features are retrieved. Differently, dynamic user behaviour is calculated out of the sport activity table. This contains all the sport activities tracked by the users on the apps. Given the size of the two tables being over 1 Terabyte (TB), querying them can be expensive in terms of time. As a solution, partitions are used within the relational table to speed up queries. A partition is composed of a subset of rows in a table that share the same value for a predefined subset of columns, called the partitioning columns. In particular, in the case of the sport activity table, the year, month, and day the activity was performed are used as partitioning columns. Given the current date and the size of the time window, partitions can be used to only query those rows include in the relevant timeframe.

The sport activity table is also queried to get all the document-related features. These indeed describe the context of the performed activity. Once again, date partitions are used for a fast data retrieval.

Finally, the features describing past interactions between the user target of the recommendations and the one who performed the activity are retrieved from another table. The interaction table contains all the social interactions happening on the news feed of the two apps. Once again, this table is also partitioned by date columns, which are used in combination to the dynamic time window to collect users' behaviour.

Static and dynamic features have to be retrieved differently. The first one can be

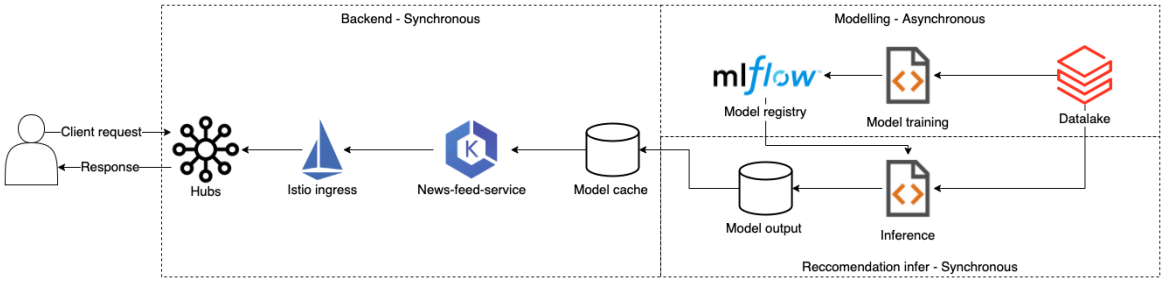


Figure 6.2: Flow diagram for synchronous recommendations.

rapidly queried from the tables without any extra computation necessary. The latter requires extra computations that involve multiple rows over the whole time window. For example, a dynamic feature counts the number of unidirectional interactions from user A to user B. Another one divides this count feature by the number of total interactions from user A to get the percentage of interactions from A to B over all his reactions. So, dynamic features' computation is not constant in time, but depends both on the size of the time window and the number of rows involved. As a solution, we want to schedule and anticipate the computation of dynamic features so that they are available at runtime for each user at the moment a request is sent from a client. Section 6.3.1 explains the followed approach.

Once the saved model and the data are available at runtime, the new activities can be ranked at each request and displayed to the users. Figure 6.2 summarizes the flow of a a news feed request from a client.

6.3.1 Feature Engineering

Computing the aggregated dynamic features at the moment a request is sent is not feasible. As a solution, we want to schedule a part of the pipeline, the one involving dynamic features. With scheduling, we mean running the pipeline prior to the request and store the results so that they can be rapidly read at the moment the request arrives.

One on hand, scheduling the feature engineering process obviously allows us to free the run of the model from the dependency of data computation. On the other hand, the model might receive as input data that is not the most recent one. Indeed, as the data pipeline is scheduled, it is run at a fixed time and its results stored. Then, the results are only updated at the next run of the scheduled process. Consequently, if, for example, a request arrives three hours after the scheduled job, the features given as input to the model will miss the last three hours of data. This delay, however, only regards the features and not the activities we are ranking. It means that at any request of a user U, all the activities recently performed by his or her friends will be

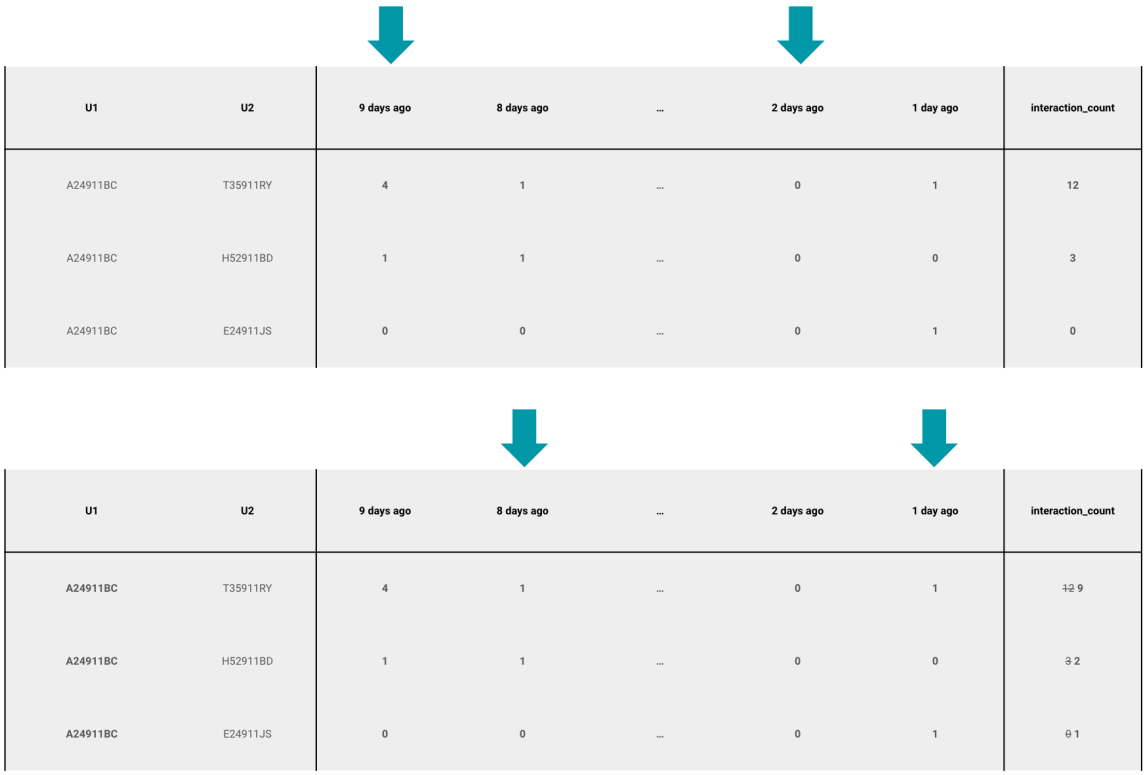


Figure 6.3: An example of the scheduled feature engineering process.

ranked, including the ones terminated a few minutes prior to the request.

Scheduling the features retrieving and storing the result allows us to sequentially update the results by only moving the dynamic window. Suppose we have a moving window of seven days and the feature engineering job is scheduled to run once a day. At every day, the window is simply moved further by one day. By doing this, data from eight days ago $T - 8$ leaves the window, while the most recent one from one day ago $T - 1$ enters the time window. Going over the data of these two days is enough to update the stored results. The results are update by removing from the computation the data of $T - 8$ and adding that of $T - 1$

Figure 6.3 shows an example. Here, the feature *interaction_count* counts the number of past interactions between U_1 and U_2. Each day column counts shows the number of interactions happened between the two users on that day. The size of the time window is fixed and at every job run it is moved one day further. At first, the data from $T - 9$ to $T - 2$ is used to calculate the result feature. For every pair of connections between two users, the result is stored in the social connection table. At the next iteration, the window is moved between $T - 8$ and $T - 1$. The count of $T - 9$ is subtracted to the results and the one of $T - 1$ added.

One main advantage of this approach is that table partitions can be used when updating the results to only read the needed portion of data. In this example, as

the job is scheduled to run once a day, we can use date partitions on the interaction table in order to rapidly remove from the count the interactions that are now eight days old and add those happened in the last twenty-four hours.

Also, this solution can handle new interactions between two users who have never interacted before. The third row of the two tables in Figure 6.3 shows such an example. To save disk space, a sparse matrix can be used and therefore no results at all would be saved for users who did not have any interaction yet. So, if a new pair of users is found when scraping the latest interactions, their features can be stored for the first time.

Chapter 7

Conclusions and recommendations

This final chapter draws the conclusion of this report. Firstly, section 7.1 states how what has been done can contribute both to Runtastic and to the diversity-related question. Then, section 7.2 goes through future improvements that can be done to help both Runtastic and future research.

7.1 Conclusions

This report showed how Runtastic's news feed could be improved by using a recommender system to sort activities in a relevant order for the user. A number of have been introduced by explaining their feasibility with respect to Runtastic's specific use case. The paper theoretically shows that personalization through content-based ranking fits best the problem compared to, for example, graph based solutions.

While the most important aspect is increasing the number of interactions on the news feed, the report showed that also increasing the number of users receiving reactions might bring benefits in terms of retention and engagement. To do that, we proposed including a measure of diversity as an important metric in both the model selection and dataset creation phases. This diversity metric was used together with other ranking metrics such as MAP, MRR, and NDCG which measures the performance of the models in different contexts. In general, the three ranking metrics followed similar trends by increasing and decreasing with respect to the model and the dataset used. Some cases showed that using a diversity-aware variation of the NDCG metric allowed some models to outperform the other because of their performances with respect to Diversity@10 despite having a worse NDCG. Pointwise and pairwise ranking on the binary labelled dataset are an example of such behaviour. Result diversification is guaranteed and improved while only giving up a slight decrease in ranking metrics. In general, including diversity within the evaluation function showed that we can optimize such trade-off between ranking and

diversity metrics.

How the datasets were created also showed to have an impact on how the models would perform. Out of the two time windows that were used to collect the dataset's features, the longer one resulted in better performance. This suggests that users tend to prefer interacting with a similar kind of content over a longer period of time, and they do not change preferences rapidly. Also, MAP and MRR displayed higher values on the datasets that were not using a binary relevance label. However, longer time windows come with a more complicated feature engineering process, which could be extremely relevant for the synchronous use cases.

Finally, we showed how scheduling feature engineering can remove the real-time dependency of the whole product on the data sources, which are slow and not suited for such a real-time use case. On the other hand, scheduling might cause losing a portion of data and consequently running the recommendation model on data that is not fully up to date. However, the results suggested how a longer time window benefits the model's performances. At the same time, a longer time window is less affected than a short one from the scheduling architecture, since the part of missing data would be less relevant compared to the longer user's past behaviour.

7.2 Recommendations

This project took a different path in order to guarantee results diversification, through the comparison of differently generated dataset and the addition of a diversity-aware metric as optimization metric. However, future work may be necessary to better understand the benefits of incorporating diversity into the training phase of the learning-to-rank models. Also, especially when it comes to pairwise and listwise approaches, more complex algorithms and models might be tried to see how this benefits the performances.

Further work can also be done on how the relevance labels are encoded. In this project, we looked at different linear combinations of likes and comments. As said earlier, more implicit feedback such as dwell time could be integrated when calculating the label. Also, diversity and coverage could be optimized by encoding into the label whether an activity has already received interactions or not. In this way, activities with no reaction would be given more importance.

Bibliography

[1] J. Brown, A. J. Broderick, and N. Lee, "Word of mouth communication within online communities: Conceptualizing the online social network," *Journal of interactive marketing*, vol. 21, no. 3, pp. 2–20, 2007.

[2] J. Stragier, P. Mechant, L. De Marez, and G. Cardon, "Computer-mediated social support for physical activity: a content analysis," *Health Education & Behavior*, vol. 45, no. 1, pp. 124–131, 2018.

[3] N. Moniz and L. Torgo, "Multi-source social feedback of online news feeds," *arXiv preprint arXiv:1801.07055*, 2018.

[4] K. N. Rao, "Application domain and functional classification of recommender systems–a survey," *DESIDOC Journal of Library & Information Technology*, vol. 28, no. 3, p. 17, 2008.

[5] J. Freyne, S. Berkovsky, E. M. Daly, and W. Geyer, "Social networking feeds: recommending items of interest," in *Proceedings of the fourth ACM conference on Recommender systems*, 2010, pp. 277–280.

[6] C. Wu, F. Wu, T. Qi, Q. Liu, X. Tian, J. Li, W. He, Y. Huang, and X. Xie, "Feedrec: News feed recommendation with various user feedbacks," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 2088–2097.

[7] M. Chegini, J. Bernard, P. Berger, A. Sourin, K. Andrews, and T. Schreck, "Interactive labelling of a multivariate dataset for supervised machine learning using linked visualisations, clustering, and active learning," *Visual Informatics*, vol. 3, no. 1, pp. 9–17, 2019.

[8] F. Ricci, L. Rokach, and B. Shapira, "Recommender systems: introduction and challenges," in *Recommender systems handbook*. Springer, 2015, pp. 1–34.

[9] K. Miyahara and M. J. Pazzani, "Improvement of collaborative filtering with the simple bayesian classifier," *Information Processing Society of Japan*, vol. 43, no. 11, 2002.

[10] R. Burke, “Hybrid web recommender systems,” *The adaptive web*, pp. 377–408, 2007.

[11] J. Lu, D. Wu, M. Mao, W. Wang, and G. Zhang, “Recommender system application developments: a survey,” *Decision Support Systems*, vol. 74, pp. 12–32, 2015.

[12] Y. Koren and R. Bell, “Advances in collaborative filtering,” *Recommender systems handbook*, pp. 77–118, 2015.

[13] M. Deshpande and G. Karypis, “Item-based top-n recommendation algorithms,” *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 143–177, 2004.

[14] X. Su and T. M. Khoshgoftaar, “A survey of collaborative filtering techniques,” *Advances in artificial intelligence*, vol. 2009, 2009.

[15] G. Linden, B. Smith, and J. York, “Amazon. com recommendations: Item-to-item collaborative filtering,” *IEEE Internet computing*, vol. 7, no. 1, pp. 76–80, 2003.

[16] T. Hofmann, “Latent semantic models for collaborative filtering,” *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 89–115, 2004.

[17] S. Ahmadian, N. Joorabloo, M. Jalili, M. Meghdadi, M. Afsharchi, and Y. Ren, “A temporal clustering approach for social recommender systems,” in *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 2018, pp. 1139–1144.

[18] Y. Shi, M. Larson, and A. Hanjalic, “Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges,” *ACM Computing Surveys (CSUR)*, vol. 47, no. 1, pp. 1–45, 2014.

[19] W. Chen, F. Cai, H. Chen, and M. D. Rijke, “Joint neural collaborative filtering for recommender systems,” *ACM Transactions on Information Systems (TOIS)*, vol. 37, no. 4, pp. 1–30, 2019.

[20] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, “Neural collaborative filtering,” in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 173–182.

[21] P. B. Thorat, R. Goudar, and S. Barve, “Survey on collaborative filtering, content-based filtering and hybrid recommendation system,” *International Journal of Computer Applications*, vol. 110, no. 4, pp. 31–36, 2015.

[22] C. Basu, H. Hirsh, W. Cohen *et al.*, "Recommendation as classification: Using social and content-based information in recommendation," in *Aaai/iaai*, 1998, pp. 714–720.

[23] M. De Gemmis, P. Lops, G. Semeraro, and P. Basile, "Integrating tags in a semantic content-based recommender," in *Proceedings of the 2008 ACM conference on Recommender systems*, 2008, pp. 163–170.

[24] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *Acm transactions on interactive intelligent systems (tiis)*, vol. 5, no. 4, pp. 1–19, 2015.

[25] Y. Liu and J. Yang, "Improving ranking-based recommendation by social information and negative similarity," *Procedia Computer Science*, vol. 55, pp. 732–740, 2015.

[26] B. Chen, X. Hu, Y. Huo, and X. Deng, "Research on recommendation method of product design scheme based on multi-way tree and learning-to-rank," *Machines*, vol. 8, no. 2, p. 30, 2020.

[27] T.-Y. Liu *et al.*, "Learning to rank for information retrieval," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 3, pp. 225–331, 2009.

[28] J. Liang, J. Hu, S. Dong, and V. Honavar, "Top-n-rank: A scalable list-wise ranking method for recommender systems," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 1052–1058.

[29] D. Cossock and T. Zhang, "Subset ranking using regression," in *International conference on computational learning theory*. Springer, 2006, pp. 605–619.

[30] A. Shashua and A. Levin, "Taxonomy of large margin principle algorithms for ordinal regression problems," *Advances in neural information processing systems*, vol. 15, pp. 937–944, 2002.

[31] P. Li, Q. Wu, and C. Burges, "Mcrank: Learning to rank using multiple classification and gradient boosting," *Advances in neural information processing systems*, vol. 20, 2007.

[32] A. Phophalia, "A survey on learning to rank (letor) approaches in information retrieval," in *2011 Nirma University International Conference on Engineering*. IEEE, 2011, pp. 1–6.

[33] Y. Freund, R. Schapire, and N. Abe, "A short introduction to boosting," *Journal-Japanese Society For Artificial Intelligence*, vol. 14, no. 771-780, p. 1612, 1999.

[34] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, “An efficient boosting algorithm for combining preferences,” *Journal of machine learning research*, vol. 4, no. Nov, pp. 933–969, 2003.

[35] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, “Learning to rank using gradient descent,” in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 89–96.

[36] C. J. Burges, “From ranknet to lambdarank to lambdamart: An overview,” *Learning*, vol. 11, no. 23-581, p. 81, 2010.

[37] M. Taylor, J. Guiver, S. Robertson, and T. Minka, “Softrank: optimizing non-smooth rank metrics,” in *Proceedings of the 2008 International Conference on Web Search and Data Mining*, 2008, pp. 77–86.

[38] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li, “Listwise approach to learning to rank: theory and algorithm,” in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 1192–1199.

[39] C. Li, H. Feng, and M. d. Rijke, “Cascading hybrid bandits: Online learning to rank for relevance and diversity,” in *Fourteenth ACM Conference on Recommender Systems*, 2020, pp. 33–42.

[40] L. Yan, Z. Qin, R. K. Pasumarthi, X. Wang, and M. Bendersky, “Diversification-aware learning to rank using distributed representation,” in *Proceedings of the Web Conference 2021*, 2021, pp. 127–136.

[41] L. Li, W. Chu, J. Langford, and R. E. Schapire, “A contextual-bandit approach to personalized news article recommendation,” in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 661–670.

[42] K. Hofmann, A. Schuth, S. Whiteson, and M. De Rijke, “Reusing historical interaction data for faster online learning to rank for ir,” in *Proceedings of the sixth ACM international conference on Web search and data mining*, 2013, pp. 183–192.

[43] A. Slivkins, F. Radlinski, and S. Gollapudi, “Learning optimally diverse rankings over large document collections,” in *ICML*, 2010.

[44] J. Wasilewski and N. Hurley, “Incorporating diversity in a learning to rank recommender system,” in *The twenty-ninth international flairs conference*, 2016.

[45] A. Ghazimatin, R. Saha Roy, and G. Weikum, “Fairy: A framework for understanding relationships between users’ actions and their social feeds,” in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019, pp. 240–248.

[46] S. Belkacem, O. Boussaid, and K. Boukhalfa, “Ranking news feed updates on social media: A comparative study of supervised models.” in *EGC*, 2020, pp. 499–506.

[47] H. Tang, J. Liu, M. Zhao, and X. Gong, “Progressive layered extraction (ple): A novel multi-task learning (mtl) model for personalized recommendations,” in *Fourteenth ACM Conference on Recommender Systems*, 2020, pp. 269–278.

[48] A. Singh and T. Joachims, “Fairness of exposure in rankings,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2219–2228.

[49] M. Vougioukas, I. Androutsopoulos, and G. Paliouras, “Identifying retweetable tweets with a personalized global classifier,” in *Proceedings of the 10th Hellenic Conference on Artificial Intelligence*, 2018, pp. 1–8.

[50] C. De Maio, G. Fenza, M. Gallo, V. Loia, and M. Parente, “Time-aware adaptive tweets ranking through deep learning,” *Future Generation Computer Systems*, vol. 93, pp. 924–932, 2019.

[51] L. Kuang, X. Tang, M. Yu, Y. Huang, and K. Guo, “A comprehensive ranking model for tweets big data in online social network,” *EURASIP Journal on Wireless Communications and Networking*, vol. 2016, no. 1, pp. 1–9, 2016.

[52] S. Berkovsky, J. Freyne, and G. Smith, “Personalized network updates: increasing social interactions and contributions in social networks,” in *International Conference on User Modeling, Adaptation, and Personalization*. Springer, 2012, pp. 1–13.

[53] P. V. Marsden and K. E. Campbell, “Measuring tie strength,” *Social forces*, vol. 63, no. 2, pp. 482–501, 1984.

[54] J. Wu and Z. Lin, “Research on customer segmentation model by clustering,” in *Proceedings of the 7th international conference on Electronic commerce*, 2005, pp. 316–318.

[55] M. Grčar, D. Mladenič, B. Fortuna, and M. Grobelnik, “Data sparsity issues in the collaborative filtering framework,” in *International workshop on knowledge discovery on the web*. Springer, 2005, pp. 58–76.

[56] X. Wang, C. Li, N. Golbandi, M. Bendersky, and M. Najork, “The lambdaloss framework for ranking metric optimization,” in *Proceedings of the 27th ACM international conference on information and knowledge management*, 2018, pp. 1313–1322.

[57] H. A. Güvenir and M. Kurtcephe, “Ranking instances by maximizing the area under roc curve,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 10, pp. 2356–2366, 2012.

[58] X. Yi, L. Hong, E. Zhong, N. N. Liu, and S. Rajan, “Beyond clicks: dwell time for personalization,” in *Proceedings of the 8th ACM Conference on Recommender systems*, 2014, pp. 113–120.

[59] N. Craswell, *Mean Reciprocal Rank*. Boston, MA: Springer US, 2009, pp. 1703–1703. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_488

[60] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. D. Cox, “Hyperopt: a python library for model selection and hyperparameter optimization,” *Computational Science & Discovery*, vol. 8, no. 1, p. 014008, 2015.

[61] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, “Algorithms for hyper-parameter optimization,” *Advances in neural information processing systems*, vol. 24, 2011.

[62] E. Bassani, “ranx: A blazing-fast python library for ranking evaluation and comparison,” in *ECIR (2)*, ser. Lecture Notes in Computer Science, vol. 13186. Springer, 2022, pp. 259–264.

[63] M. D. Smucker, J. Allan, and B. Carterette, “A comparison of statistical significance tests for information retrieval evaluation,” in *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, 2007, pp. 623–632.

[64] D. Basu*, “Randomization analysis of experimental data: the fisher randomization test,” *Selected Works of Debabrata Basu*, pp. 305–325, 2011.

[65] M. J. Moshkov, “Time complexity of decision trees,” in *Transactions on Rough Sets III*. Springer, 2005, pp. 244–459.

[66] J. Singh, “Computational complexity and analysis of supervised machine learning algorithms,” in *Next Generation of Internet of Things*. Springer, 2023, pp. 195–206.

[67] Z. Sun, G. Pedretti, P. Mannocci, E. Ambrosi, A. Bricalli, and D. Ielmini, “Time complexity of in-memory solution of linear systems,” *IEEE Transactions on Electron Devices*, vol. 67, no. 7, pp. 2945–2951, 2020.

[68] P. Elger and E. Shanaghy, *AI as a Service: Serverless machine learning with AWS*. Manning Publications, 2020.