Explaining Rankings

Author:
Maartje Anne TER HOEVE
maartje.terhoeve@student.uva.nl
10190015

Supervisor:
Prof. Dr. Maarten de RIJKE
derijke@uva.nl

September 15, 2017
Abstract

Machine learning algorithms have become more complex over time and therefore it has become more difficult to understand the underlying decisions of these algorithms. In this research we investigate the explainability of ranking algorithms. In particular we focus on the ranking algorithm of Blendle, an online news kiosk that uses a ranking algorithm to make a personalized selection of news articles from a wide variety of newspapers for their users.

From a user study on 541 Blendle users we learn that users would like to receive explanations for their personalized news ranking, however, they do not show a clear preference as to how these explanations should be shown. Supported by these results we design LISTEN, a model-agnostic LISTwise EXplanatIon method to explain the decisions of any ranking algorithm. Our method is model-agnostic, because it can be used to explain any ranking algorithm without the need to add additional information about the specific algorithm. Our method is listwise, because it explains the importance of features to the ranking by taking the influence of the features on the entire ranking into account. For rankings, existing pointwise approaches, where the importance of a feature is calculated by only looking at its influence on the item score, are not faithful. This is because the position of an item in the ranking is not only defined by its own score, but also by the score of the other items in the ranking. The new listwise approach is an important contribution of this work. The importance of features is found by gradually changing feature values and computing the effect on the entire ranking. The main intuition behind this approach is that if perturbations to features are able to change the ranking a lot, these are important features. If the perturbation of a feature does not change the ranking, this feature is not important for the ranking.

In order to allow our explanation model to run in production, where it needs to compute explanations for news articles on the fly, we implement two steps to increase the speed. First we divide the process of changing the feature values in two parts. In the first part we find the most disruptive feature values. In the second part we use only these values to find the most important features. As a second speed up, we train a neural network on the data that we made in the previous step. In production we only use the neural network to compute the explanations. We call this method Q-LISTEN. This speed up is another important contribution of this work.

We compare LISTEN and Q-LISTEN with two baselines: the already existing Blendle reasons (these are heuristic and therefore unfaithful explanations of the underlying ranking algorithm) and the reasons produced by LIME (Ribeiro et al., 2016), a local, pointwise explanation method. An offline evaluation shows that LISTEN produces faithful explanations and that the two speed up steps barely decrease the accuracy of the model. A large-scale online evaluation on all Blendle users who receive a personalized news selection shows that the type of explanation does not influence the number of reads of the users, which indicates that even though users find it important to receive explanations, they are less sensitive to the faithfulness of these explanations.
Acknowledgments

Maartje Anne ter Hoeve, September 15, 2017

Some people deserve a special thank you.

Maarten, for being my supervisor. I learned a lot from you, contentwise, but also about how to structure my thoughts and work. Evangelos, for being my assessor and for making me enthusiastic about Information Retrieval in the first place.

The Blendle people. Anne, I could not have wished for a better supervisor outside university. I liked how we could vividly discuss, well, practically anything. Daan, for always being willing to talk through ideas. Jeffrey, for reading my entire thesis and giving feedback. Koen and Arno, for taking the time to go through my way too long PRs. Lucas, for discussing our thesis topics together and getting inspiration from that. Martijn, for always being supportive. Mathieu, for knowing everything about Looker.

Jörg, Maurits, Thijs, I liked how we could spend hours on a few square meters, with nothing more than our four laptops and Maurits playing random songs out of nowhere. How we became friends.

Papa, mama, for supporting me in the decisions I made. Papa, for going through the first chapter of a math book with me, teaching me about spheres and rectangles, when I was only five. Mama, for helping me with chemistry and showing me that truly any topic could be fun. You both raised me in such a way that I have never felt any limitations to learn something new. Jaco, for being the best brother I could ever imagine.

Thank you.
6 Method to explain rankings: (Q-)LISTEN

6.1 RQ 3 - How do we provide users with understandable, uncluttered
listwise explanations? .................................................. 40

6.2 LISTEN: a LISTwise ExplaiNer ................................. 42
  6.2.1 Training phase ................................................... 42
  6.2.2 Explaining phase ............................................... 45

6.3 Q-LISTEN: Speed ups with neural networks .................. 46

6.4 Dealing with diversification ...................................... 48

6.5 Communication to the user ...................................... 49

7 Experimental setup ..................................................... 52

7.1 Data ................................................................. 52

7.2 mLIME baseline .................................................... 52

7.3 LISTEN ............................................................ 54

7.4 Q-mLIME and Q-LISTEN ......................................... 57

7.5 Evaluation .......................................................... 58

8 Results ........................................................................ 62

8.1 RQ 4 - Are our explanations faithful and is the method scalable? 62
  8.1.1 Are our explanations faithful? .............................. 62
  8.1.2 Is our method scalable? ....................................... 68

8.2 Some examples ....................................................... 68

8.3 RQ 5 - How do users interact with different reason types? ..... 73

9 Discussion and conclusion ............................................ 79

9.1 Answers to research questions .................................... 79

9.2 Theoretical and practical implications .......................... 81

9.3 Limitations and future work ...................................... 81

Bibliography ............................................................... 84

A User study .............................................................. 90
Chapter 1

Introduction

Machine learning algorithms become more powerful and more complex (Bengio et al., 2009; Schmidhuber, 2015). This complexity comes at a price: the algorithms also become more black boxed, causing a decreasing interpretability of the decision process of the algorithm (e.g. Adebayo and Kagal, 2016; Zafar et al., 2017). Even though we may know and understand the exact underlying structure of the algorithm and we may know how it learns and which calculations are made to come to a certain outcome, we increasingly lack the means to answer this one question: why does the algorithm behave the way it does? Our algorithms have learned to find structures in unstructured data; structures that we were not able to find ourselves and that may not even mean anything to us. So why are certain features weighted more than other features? Which properties of the data are used to come to the output? In many cases, we simply do not know, which can harm the decision making process (Pedreshi et al., 2008).

There are two main reasons why it is important to try to unravel these black boxes the algorithms have become. First of all, there are the users of the system. An increasing amount of research is dedicated to automated decision making in law, recommender systems, health care, etc. (e.g. Christin et al., 2015; Ciresan et al., 2012; Covington et al., 2016; Glocker et al., 2012; Karlsson, 2011). Users need to be able to trust the outcome of these systems. Imagine a doctor, using an artificial assistant when judging X-rays. Our A.I. classifier may label an X-ray as a positive example of a particular illness. The doctor may question this decision, as he or she does not see a reason to classify this X-ray as such. Now, if the system can explain itself, the doctor may decide whether or not to trust its decision. If the system points at a part of the picture that is indeed an indication of the particular illness, the doctor can decide to trust the system. However, if the system points at a flaw on the X-ray that the doctor knows is caused by for example a piece of dust on the camera, the doctor can decide that this decision of the system is incorrect. This also gives the doctor the opportunity to give feedback to the system. The system can learn from this feedback and adapt its future decisions accordingly.

Secondly, not only the user of the system, also the developer of the system can benefit from a system that can explain its decisions. By finding out the reasons behind the outcome of the system, a developer can gain insights in whether the system works the way it is supposed to work or whether it bases its decisions
on for example patterns in the data we know it should not base its decisions on. A famous example, given by Dreyfus and Dreyfus (1992), describes a case where a neural network was trained for the U.S. military army. The network was supposed to be able to recognize tanks that were hidden in the woods. In order to do so, the network was trained on pictures of tanks in the woods and on pictures of the woods without tanks. On both the train and the test set the network performed very well. Yet after that, the network was shown new pictures and it performed extremely bad. It was only after a while that one of the developers found out that all pictures without tanks had a cloudy sky, whereas all pictures with tanks on it had a splendid sunny sky. The system had not learned to distinguish between “tank” and “no tank” but between “sunny” and “cloudy”. Now, we have learned our lessons from this mistake and nowadays we always carefully construct a training and test set that reflect the patterns in the real world as close as possible — especially when we design systems that are to be used in the real world. Yet one can imagine that there can be other patterns in the data that are less apparent, but that we do not want to base our decisions on either.

The question of unraveling the black box of machine learning and (later) deep learning algorithms has been around for a long time (e.g. Bilgic and Mooney, 2005; Hendricks et al., 2016; Herlocker et al., 2000; Tintarev, 2007), yet has become very relevant at the time of writing (mid 2017). Not only is the research community very interested in the topic, also the European Union has approved the General Data Protection Regulation (GDPR) on April 14, 2016. The GDPR will be enforced on May 25, 2018, and states, amongst others, that algorithmic decisions need to be explainable.

In Chapter 2 we give an extensive overview of the research that is done on explainability of machine learning algorithms. Not much work has been dedicated to the explainability of rankings. In this study we design LISTEN, a method to explain a ranking produced by any type of ranking algorithm. LISTEN stands for LISTwise EXPLAIINer. The general goal of a ranking algorithm is to order a set of items based on their relevance. Determining this relevance is part of the ranking algorithm’s job too. A Search Engine Result Page (i.e., the page that is returned after entering a query to a search engine), also known as SERP, is a well-known application of a ranking algorithm. In order to make this page, the relevance scores of the web pages need to be computed and the pages need to be returned to the user in decreasing order of relevance. Other applications where ranking algorithms are used are shopping websites and recommender systems (systems that are used to automatically recommend items to users, for example movies and series on Netflix). In order to allow LISTEN to run in real time we extend it by training a neural network on input and output data generated by LISTEN. We call this extension Q-LISTEN. We test our findings on the ranking algorithm that is used for the the recommender system of Blendle, a Dutch start-up that serves as an online news kiosk.

1.1 What are explanations?

In the social sciences there has been a lot of research conducted on explanations. Miller (2017) gives an extensive overview of those studies and how they
could be of use for generating explanations in artificial intelligence. Miller et al. (2017) summarize some of the main findings of this work. We use both studies to define the notion of explanation in the current research. In general, an explanation gives the cause of why something happened. This can be expressed in multiple ways, for example textual, but also visual. Four other properties of good explanations are stated to be “quality”, “quantity”, “relation” and “manner”, which respectively mean that one should aim for truthful explanations, that include as much information as is needed (not more), one should only include relevant information and one should phrase it in a polite way. With these studies in mind, it is our goal to generate explanations that obey to these four properties and that give the main cause of why the ranking is as it is and, in particular, the main cause of the appearance of an item at its position in the ranking.

At this stage we also need to look into the notion of interpretability. The most precise cause of an event will be the precise underlying mathematical structure of the algorithm and the precise calculations that are made, yet in most cases this is not understandable (not even by experts in the field). Doshi-Velez and Kim (2017) define interpretability as: “the ability to explain or to present in understandable terms to a human”. Now, “understandable to a human” is still somewhat vague. It is not the goal of this research to automatically generate understandable text or anything comparable that could serve as an explanation. Instead, we aim to find the most important causes of an event that can be directly mapped to a human understandable message.

We want our explanation model to be model-agnostic (given that we are explaining rankings) and faithful. By model-agnostic we mean that our model should be able to explain any type of ranking algorithm. By faithful we mean that explanations should truthfully describe the main cause of an event by looking at the underlying algorithm. In this sense faithfulness is linked to the quality property that was mentioned before. Creating faithful explanations is an important motivation to conduct this research on explaining rankings, as the explanation of a ranked list, or a SERP, differs from explaining single individual data points, e.g. single recommendations in the context of recommender systems. Whereas for the latter it suffices to only look at the properties of the item and the user (and potentially the properties of other users that this item was recommended to as well), for a faithful explanation of a ranked list all elements in that list need to be taken into consideration. In the next section we look into approaches to explain rankings in more detail.

1.2 Approaches to explain rankings

Imagine a ranking algorithm that uses a simple linear ranking scoring function to compute the relevance of particular items. The ranking function is given by

\[ \text{score}(x_0, x_1, x_2) = 0.2x_0 + 0.3x_1 + 0.5x_2, \]  

where \( x_0, x_1 \) and \( x_2 \) are features. In a real application these could be features that describe characteristics of the item, the user, general features such as the current season or time, etc. However, for the current example we will just stick with the abstract notion of ‘features’, without worrying what these features
represent. \( x_0 \) and \( x_1 \) can take on values in the range \([0, 1]\) and \( x_2 \) can take on values in the range \([0.6, 1]\). Also, imagine that we have a ranking with three items that are described by the feature value matrix

\[
\begin{array}{ccc|c}
\text{document} & x_0 & x_1 & x_2 & \text{score} \\
\hline
d_0 & 1 & 1 & 1 & 1 \\
d_1 & 0.5 & 0.5 & 1 & 0.75 \\
d_2 & 1 & 0 & 0.7 & 0.55 \\
\end{array}
\]

where the last column is the score computed by Equation 1.1 and \( d \) stands for document.

Our task is to explain this ranking. There are at least two approaches we could take. We could focus on a single document and its corresponding score and mark the feature that contributed most to the score as the most important feature and hence, give this feature as explanation for why this document is selected for this ranking. This is what we call a pointwise explanation, because it only takes one item, i.e. one point, in the ranking into account when explaining the occurrence of that item in the ranking. One important shortcoming of this approach is that it does not explain the rank of a particular item, it just explains its score. In order to explain the rank of an item, one needs to take the other items in the ranking into account as well. This is what we call the listwise approach, because this approach looks at the entire list of items for its explanations. Below we give an example to show the difference between the two approaches: the pointwise approach on the one hand and the listwise approach on the other hand.

We use the feature value matrix that we introduced above and we want to find the most important feature for the first item in the ranking, \( d_0 \). A pointwise approach would mark feature \( x_2 \) as most important, as this feature value, together with its corresponding weight, makes the score go up most. A listwise explanation would mark feature \( x_1 \) as the most important feature, something a pointwise explanation would never do, as \( 0.3 \times 0 = 0 \). However, a listwise explanation would find that feature \( x_2 \) is not able to change the ranking. Changing it to the largest possible value, 1, would make the score become 0.7 and changing \( x_2 \) to its lowest possible value would make the score be 0.45, which both leave the ranking as it is. On the other hand, changing \( x_1 \) to 1, would give \( d_2 \) the second position in the ranking, above \( d_1 \) as then the score would become 0.85.

These two toy examples show that a pointwise explanation method does not always capture the behaviour that we want to explain. Moreover, many-state-of-the-art ranking algorithms are optimized to learn an entire ranking, instead of
individual scores of items in a ranking. Therefore, listwise explanations can be said to provide more faithful explanations of the ranking. Now we just looked at two toy examples, but similar reasoning holds for more complex scoring functions.

In this research we aim to develop a listwise explanation method, as opposed to a pointwise explanation method. In the context of ranking algorithms, we define a pointwise explanation as an explanation that only takes the score of an individual item into account. We define a listwise explanation as an explanation that takes the entire ranking into account. What a listwise explanation could look like in practice is one of our research questions. We address this question in Section 6.1. As stated, we develop and test our system on the Blendle recommender system. Because of that, we briefly look into Blendle first, before we list our research questions and contributions.

1.3 What is Blendle?

Blendle is a Dutch start-up that serves as an online news kiosk and is backed by amongst others the New York Times. At the time of writing Blendle has over a million users. Every day, Blendle users receive a personalized selection of news articles, selected based on a number of features that capture their reading behaviour and topical interests. On top of this, Blendle users also receive a number of must reads every day; these articles are selected by Blendle’s editorial staff and are the same for everyone. This is one of the ways to prevent users ending up in their own filter bubble. Blendle allows users to purchase a single news article instead of having to buy an entire newspaper (using micropayments) or to prepay for all articles in their personal selection via a subscription model (called Blendle Premium). Users have the possibility to receive a refund for an article if they are not satisfied with it. A user’s personal selection of news articles is composed by a ranking algorithm and this makes the Blendle environment especially suited to develop and test our listwise explanation method. The precise details of the ranking algorithm are described in Chapter 2.

1.4 Research questions and contributions

In this research we aim to develop a faithful, listwise and model-agnostic explanation method. However, before we do that we need to understand whether it is valid to assume that users would like to receive such explanations of why they see articles in their personal selection.

This research can be split into two parts. The first part investigates the users’ thoughts and opinions about explanations. The second part focusses on the design and implementation of an explanation system for ranking algorithms. We address the following research questions:

Part 1

RQ1 Do users want to receive explanations of why particular news items are recommended to them?
RQ2 What way of showing news recommendation reasons do users prefer: textual or visual reasons; a single reason or multiple reasons; apparent or less apparent reasons?

Part 2

RQ3 How do we provide users with easy to understand, uncluttered, listwise explanations?

RQ4 How do we build an explanation system that produces faithful explanations for the outcome of a ranking algorithm, yet is scalable so that it can run in real time?

RQ5 Does the reading behaviour of users who are provided with model-agnostic listwise explanations for a personalized ranked selection of news articles differ from the reading behaviour of users who are provided with heuristic or pointwise explanations for a personalized ranked selection of news articles?

In answering these research questions, our findings contribute to how recommendations should be presented and, more broadly, to our understanding of how listwise explainability can be operationalized. Our listwise approach contributes a faithful approach for explaining ranking algorithms. Moreover, we contribute an explanation pipeline that can run in real time and that can therefore be used in real life applications.
Chapter 2

Related work

This chapter gives an overview of the related work that has been done on explainability so far and that is relevant for the current study. In the first part of this chapter, we examine the notion of explanation further. We briefly started this in Section 1.1. In the second part of this chapter, we present a number of studies that serve as examples of studies that investigate the explainability of machine learning algorithms.

2.1 Explanations: What and Why?

In Section 1.1 we referred to research by Miller (2017), Miller et al. (2017) and Doshi-Velez and Kim (2017) to define the concept of explanation as we use it throughout this study. We defined the goal of an explanation to give the main cause or causes of why an event happened. Moreover, we want our explanations to be faithful and model-agnostic. Related to these last two characteristics we look into the distinction between justifications and descriptions as explanation methods (Vig et al., 2009) and between black box approaches and blind box approaches (Hosseini et al., 2017).

Justifications and descriptions Vig et al. (2009) introduce two kinds of explanation styles: justifications on the one hand and descriptions on the other hand. Justifications focus on providing conceptual explanations that do not necessarily expose the underlying structure of the algorithm, whereas descriptions are meant to do exactly that. In this work, we aim to provide descriptions instead of justifications, as one of our main goals is to provide faithful explanations. Descriptions can be “local” or “global”. Local descriptions only explain or simulate the underlying structure of a particular part of the model, whereas global descriptions aim to explain the entire model, thereby not allowing for simplifications of the model by only looking at a particular part of the model.

Black boxes and blind boxes Up until now we have used the term black box for an algorithm whose outputs are difficult to explain. However, it is worth being more specific on this terminology. Hosseini et al. (2017) distinguish between
black boxes and blind boxes:

- **The black box.** We can use the underlying algorithm that is the subject of the explanation as an oracle. That is, we can feed input to the model and receive its output. We cannot access anything else than the model’s input and output.

- **The blind box.** We cannot use the underlying model as an oracle. That means that we cannot query the underlying algorithm for predictions, given some input. We only know that there is an algorithm that we need to explain.

A third approach, that is not mentioned by Hosseini et al., that we could add to this list is a **white box approach**, in which we not only know the input and the output of the model, but also the precise steps that are made to come to this output. Note that Hosseini et al. do not use the two mentioned approaches to make explanations, but to block the transferability of adversarial examples, which is another field of research. In the current study we use the black box approach, as one of the aims of this research is to design an explaining algorithm that is model-agnostic, but we are willing to use the model’s input and output.

**Contrastive explanations**  Miller (2017) and Miller et al. (2017) mention the notion of *contrastive explanation*, that states why event A happens rather than event B. An example could be why a certain image was labeled as a train instead of a car. In the setting of the current research a contrastive explanation could be why an item was ranked above another item or why an item occurs in the ranking whereas another item does not. As our third research question we investigate how we can explain a ranked list. In Section 6.1 we answer this research question.

**Motivations for explanations**  Tintarev (2007) lists seven possible aims when explaining the outcomes of an algorithm to users: *transparency, scrutability, trust, effectiveness, persuasiveness, efficiency and satisfaction*. In Table 2.1 these aims are listed, together with an explanation. These aims are related to the four properties of good explanations by Miller et al. (2017) that we listed in Section 1.1: *quality, quantity, relation and manner*. Herlocker et al. (2000) also list four main motivations to provide users with explanations: *justifications, user involvement, education and acceptance*. These motivations are listed in Table 2.2 together with a brief description. Several studies have shown that adding explanations contributes positively to one or more of these goals (e.g. Bilgic and Mooney, 2005; Dzindolet et al., 2003; Hendricks et al., 2016; Herlocker et al., 2000; Musto et al., 2016; Pu and Chen, 2007; Ribeiro et al., 2016).

In what follows we have a closer look into previous work on the explainability of machine learning algorithms.
Table 2.1: Explanation aims and their meanings by Tintarev (2007).

<table>
<thead>
<tr>
<th>Metric</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transparency</td>
<td>Does the user understand the explanation?</td>
</tr>
<tr>
<td>Scrutability</td>
<td>Make sure users can state that the explanation is incorrect</td>
</tr>
<tr>
<td>Trust</td>
<td>The explanation causes trust in the algorithm that was used</td>
</tr>
<tr>
<td>Effectiveness</td>
<td>Helps users to make the right decision</td>
</tr>
<tr>
<td>Persuasiveness</td>
<td>Convince users to read the article</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Helps users to make decisions faster</td>
</tr>
<tr>
<td>Satisfaction</td>
<td>Increases the user satisfaction</td>
</tr>
</tbody>
</table>

Table 2.2: Explanation motivations and their meanings by Herlocker et al. (2000).

<table>
<thead>
<tr>
<th>Metric</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Justification</td>
<td>Explanations help a user to decide whether or not to trust the recommendation.</td>
</tr>
<tr>
<td>User involvement</td>
<td>Explanations give users the opportunity to interact with the recommendation engine and to provide feedback as the user understands better why certain items are suggested.</td>
</tr>
<tr>
<td>Education</td>
<td>Explanations teach a user the benefits and the fallbacks of the system.</td>
</tr>
<tr>
<td>Acceptance</td>
<td>Explanations help users to accept the system as it is, being an assistant of the user.</td>
</tr>
</tbody>
</table>

2.2 Explanations for machine learning algorithms

In this section we look into studies that have focussed on the explainability of machine learning algorithms in general. Many studies have been conducted from a Human Computer Interaction angle (e.g. Bilgic and Mooney, 2005; Herlocker et al., 2000; Tintarev, 2007). That is, questions are asked such as “how do users interact with the system and how can explanations help with this?”. Yet these studies do not focus on constructing faithful explanations to describe the underlying decisions of the algorithm. Instead, explanations are made up to give users an idea of what the explanations could be like. Other studies do focus on faithfully describing (parts) of the underlying algorithm (e.g. Musto et al., 2016; Vig et al., 2009). Some studies focus on both sides (e.g. Pu and Chen, 2007).

Slightly differently, Abdollahi and Nasraoui (2016) design a Restricted Boltzmann Machine that recommends only those items that are explainable and Muhammad et al. (2015) directly use explanations to rank hotel recommendations. These explanations are also shown to the users. Muhammad et al. use the other items in the ranking in their explanations, that is, they construct explanations such as “this hotel has a free parking spot and is therefore better than 90% of the alternatives”. As such, one could state that this research comes close to our own aim of providing listwise explanations. However, a fundamental difference is that we aim to develop a method that can be used for any ranking algorithm, whereas Muhammad et al. come up with explanations that are
specifically designed for the specific recommendation engine — the explanations are even used to make the engine work. This makes this explanation system not model-agnostic and therefore not suitable to explain the decisions of any ranking algorithm.

Herlocker et al. investigate the addition of explanations to the recommender system of MovieLens. MovieLens uses collaborative filtering as its recommendation technique. Collaborative filtering is a technique that uses information from other users to construct the recommendation. Explanations could be of the form “Other users like you also like X”. Collaborative filtering has been proven to be difficult to use for news recommendations (the problem setting of the current research) due to what is known as the cold start or first rater problem (Melville et al., 2002; Vozalis and Margaritis, 2003). A news article needs to be recommended right after its release. At that moment the article has not been read yet and for this reason no information that can be used for collaborative filtering is known yet. Herlocker et al. investigate how explanations should be presented to the user. One could choose to use different designs, but also a variety of reasons. The authors test this by providing users of MovieLens with recommendations and explanations for these recommendations. The explanations were not faithful to the model, yet acceptable. I.e., they were manually constructed in such a way that one could believe this was the reason that this particular recommendation was shown. Users were asked how likely it was that they would select this movie on MovieLens. Users liked a histogram that showed how neighbouring users had ranked this particular movie best. A user study that was conducted revealed that the majority of the users (86%) would like to receive explanations about why particular movies were recommended to them.

Hernando et al. (2013) also aim to provide recommendation reasons for the MovieLens system. Instead of providing explanations for a single recommended item, they try to give global, visual explanation that shows the user a graph that shows all items that were recommended to this user and how these items connect to other items. Figure 2.1 shows an example of such an explanation. The authors do not report on any user studies, yet it is to be questioned whether this explanation is very intuitive for most users. This way of constructing explanations already comes closer to describing the underlying structure of the algorithm. In the remaining part of this chapter we describe several studies that aim to do generate these descriptions, in the sense of Vig et al. (2009), as well as studies that design machine learning algorithms that predict and explain these predictions in parallel.

**LIME** Ribeiro et al. (2016) introduce LIME, a method that can be used to locally explain the classifications of any classifier. LIME is used as a baseline in the current research. Three important characteristics lay at the base of the construction of LIME: an explaining model needs to be (1) “interpretable”, (2) “locally faithful” and (3) “model-agnostic”, which Ribeiro et al. respectively define as (1) “provide qualitative understanding between the input variables and the response”, (2) the explanation “must correspond to how the model behaves in the vicinity of the instance being predicted” and (3) “the explanation should be able to explain any model”. Ribeiro et al. give linear models, decision trees and falling rule lists as examples of interpretable models. LIME is minimizing
\[ \xi(x) = \arg\min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g), \quad (2.1) \]

in which \( g \) is an interpretable model in the set of interpretable models \( G \). \( \mathcal{L} \) is a loss function, that takes the original model \( f \), \( g \) and a distance metric \( \pi_x \) as input. \( \Omega(g) \) is a complexity measure for model \( g \), i.e. less complex models are preferred over more complex models. Measures of complexity can be the depth of a decision tree, the number of zero weights in a linear model, etc.

As LIME aims to be model-agnostic, data points are sampled around the data point that is to be explained. These sampled data points are classified, both by \( f \) and by \( g \). Data points are weighted by \( \pi_x \), i.e. the nearer the sampled data point to the original data point, the more important this classification is. Equation 2.2 is reflecting this. \( z \) and \( z' \) represent samples instead of ‘real’ data points. Each data point is converted to an interpretable data point. An example of interpretable data points is the use of one-hot vectors instead of word embeddings when representing words in sentences. The apostrophe in \( z' \) is used to represent interpretable data points. \( \pi_x \) is an exponential kernel, which is given in Equation 3.7 (in which \( D \) is some distance function, such as the cosine distance for text).

\[ \mathcal{L}(f, g, \pi_x) = \sum_{z, z' \in Z} \pi_x(z)(f(z) - g(z'))^2 \quad (2.2) \]

\[ \pi_x(z) = \exp\left(-\frac{D(x, z)^2}{\sigma^2}\right) \quad (2.3) \]

Based on the model that is chosen as being the best explaining and less complex model, the most important features are given as explanation for this data point. E.g. imagine that a linear model was chosen as interpretable model, the features that received the highest weights are the explanations of the model. The fact that a linear model (or any other interpretable model) is constructed around the data point that is to be explained makes LIME “locally faithful”. That is, LIME is able to simulate the local behaviour of a classifier, yet not the global behaviour. Ribeiro et al. state that one needs around 5000 sampled data points to explain a random forest. This makes LIME extremely time consuming.
In Chapter 3 the precise implementation details of LIME are unraveled, together with a description of how we use LIME as the baseline in this research.

**Predicting and explaining in parallel** Several studies train a model that predicts and explains its decisions at the same time. For example Hendricks et al. (2016) describe a method to jointly train an image classifier and an explanation system for this classifier. The system automatically generates sentences with explanations such as the example they give for an image of a Western Grebe (a particular type of bird): “This is a Western Grebe because this bird has a long white neck, pointy yellow beak and red eye”. These sentences are generated by an LSTM (Hochreiter and Schmidhuber, 1997) and provide, what Hendricks et al. call “discriminative features” of the image. The approach they take is very comparable to standard image caption generation (e.g. Donahue et al., 2015). However, Hendricks et al. add the category that is predicted by the model as input to the “caption generation module”. Moreover, they minimize a combination of two types of losses: a relevance loss and a discriminative loss. The first is used for the image caption generation and the latter uses a reinforcement learning paradigm. They show that their approach works fairly well, producing sentences such as the example given above.

Al-Shedivat et al. (2017) introduce Contextual Explaining Networks, abbreviated as CENs. These networks’ predictions and explanations go hand in hand. The explanations are “context-specific” and the corresponding model that is trained is given by the predictive distribution

\[ Y \sim p(Y|X, \theta), \quad \theta \sim p_w(\theta|C), \quad p_w(Y|X, C) = \int p(Y|X, \theta)p_w(\theta|C)d\theta, \]

(2.4)

in which \( C \in \mathcal{C} \) is the context of the model, \( X \in \mathcal{X} \) are the attributes of the model and \( Y \in \mathcal{Y} \) are the labels of the model. \( p(Y|X, \theta) \) are said to be explanations (also called hypotheses) of the model as this probability relates the attributes \( X \) to the labels \( Y \). The fact that \( p(Y|X, \theta) \) is parameterized by \( \theta \) makes the model context-specific. \( p_w(\theta|C) \) is a neural network. Al-Shedivat et al. present several formats of the precise lay-out of this network. \( \theta \) is seen as the actual explanation. They show that the explanations that are generated are very close the the explanations that are generated by LIME (Ribeiro et al., 2016).

Even though models that can explain themselves may be a desirable direction for future model designs, not all models have this property (yet) and therefore it is important to design other explanation methods as well.

**Using gradients to define importance** An intuitive way to compute feature importance is by taking the gradients of the output probability of the model with respect to the input. This idea is described by Hechtlinger (2016) and applied by Ross et al. (2017). The assumption is that if gradients are large, the features that belong to these gradients are important for this model output. Ross et al. use this idea to constrain the gradients in such a way that they match domain knowledge of which features should be important in making a certain decision. One important prerequisite of using this method is that the models are differentiable with respect to their inputs. This is a desirable model property yet
not a given one. For example the LambdaMart ranking algorithm, described in Section 3.1 and state-of-the-art these days, does not have this property. Moreover, simply taking the gradient of a scoring function yields some undesirable properties in some cases. To show this, we again use the simple scoring function that we also used in Chapter 1 and that we repeat here, given by

$$\text{score}(x_0, x_1, x_2) = 0.2x_0 + 0.3x_1 + 0.5x_2. \quad (2.5)$$

If we simply took the derivative with respect to the inputs of this model, we would only use the weights to determine feature importance, whereas we would prefer a combination of weights and feature values. Therefore this method cannot be used if one wants to make a model-agnostic explainer that can be used for any type of (ranking) model.

**Feature selection and feature importance**  The goal of feature selection is to find a relevant subset of features for a model. There is a substantial amount of research on this topic (e.g. Battiti, 1994; Dash and Liu, 1997; Geng et al., 2007; Hua et al., 2010; Lai et al., 2013; Laporte et al., 2014). Many of these studies aim to find the set of features that maximize the importance of the features in the set and minimize the similarity of features in the set. Finding the importance scores for features is related to the explainability question that we try to solve in the current research. Battiti (1994) uses Shannon’s entropy (Shannon et al., 1951) to select new features for classification problems. Features that contain most information and therefore decrease the uncertainty about a classification are selected. Several studies use dimensionality reduction techniques such as PCA for feature selection (Malhi and Gao, 2004; Yu and Liu, 2003). Geng et al. (2007) design a feature selection method for ranking. They measure the importance of features by metrics such as MAP, NDCG and loss functions such as pairwise ranking errors. Similarity between two features is measured by measuring how similar the rankings are that these two features produce. Hua et al. (2010) compute feature similarity in the same fashion. After that, they cluster features based on their similarity scores. Only a single feature from each cluster is selected.

**Skater**  Mid 2017 Skater\(^1\) was released. Skater is a Python package that can be used to make model-agnostic explanations. Skater provides the code to make both local and global explanations. For local explanations, LIME is used. For the global explanations Skater uses a similar intuition as we use in this research, namely we change feature values and feature values that generate a large change in score are assumed to be important for this particular instance. Our work is different in the following important ways: even though Skater may provide a global explanation, this is not a listwise explanation yet, as the explanation is based solely on an individual item (and not for example on other items in the ranking). Moreover, Skater uses, just like LIME, many samples to provide an explanation for a data point and is therefore not expected to be fast enough to run in a production environment.

\(^1\)https://github.com/datascienceinc/Skater

13
Chapter 3

Technical Background

In this chapter we look into the technical background of approaches that we use throughout this research. We start with ranking algorithms and the approaches that have been conducted to solve the ranking problem over time. Moreover, we describe evaluation techniques that are used in Information Retrieval. The evaluation of explanation systems is not trivial, as in the end the main reason of wanting explanations for a system, is that we do not know why the system makes certain decisions. We also describe the technical details of LIME (Ribeiro et al., 2016), that we use as one of the baselines in our research. We conclude this chapter with a brief overview on neural networks.

3.1 Ranking algorithms

Ranking is a widely studied topic that finds its applications in several domains (e.g. Del Corso et al., 2005; Haveliwala, 2003; Page et al., 1999), ranging from building search engine result pages, where a user has a specific query for the search engine, to domains in which a user has a less specific query yet is expecting to see results, such as the timelines on social networks, or the personalized selection of news that is the problem setting of the current research. Making good ranking algorithms is the aim of the Learning to Rank research.

Over the course of time several approaches to Learning to Rank have been conducted. These can be divided into pointwise approaches, pairwise approaches and listwise approaches (Liu et al., 2009). Pointwise approaches compute a relevance score for every single item that is to be ranked individually. The items are then ranked in a decreasing order of scores. Pairwise approaches look for disordered pairs in a ranking, put them in the correct order, until all pairs are ranked correctly, and thus the entire ranking as well. Listwise approaches try to optimize the order of the entire list at once and have information retrieval measures such as NDCG as the optimization objective.

Pointwise Learning to Rank  An example of a pointwise learning to rank algorithm is a log-linear model. At the time of writing this log-linear model is also used at Blendle and is given by
\[ s(w, f) = \sum_i w_i \log(f_i), \quad (3.1) \]
in which \(w_i\) is a weight that is computed for each feature value \(f_i\).

**Pairwise Learning to Rank** A well-known example of a pairwise learning to rank algorithm is *RankNet* (Burges, 2010). We do not use pairwise ranking methods in this study, yet we do mention the approach here for completeness. RankNet trains a neural network that computes the target probability that a document \(i\) is ranked above a document \(j\), based on input feature vectors \(x_i\) and \(x_j\), given a query. This target probability is given by

\[ P_{i,j} = \frac{1}{1 + \exp(f(x_i) - f(x_j))} \quad (3.2) \]

and the corresponding cross-entropy loss function that is optimized is given by

\[ C = -P_{ij} \log P_{ij} - (1 - P_{ij}) \log(1 - P_{ij}). \quad (3.3) \]

The weights of the networks are tuned by an algorithm that is very comparable to the backpropagation algorithm that is often used to train the weights in a neural network (see Section 3.4). Namely, first one ranks all items using the neural network. Then, by computing the derivatives of the loss function with respect to the input, so called \(\lambda\)-values arise. One does this for all document pairs in the ranking and one aggregates these \(\lambda\)-values. Furthermore, for each document pair, one takes the derivative of the score, i.e. the target probability with respect to the weights. The weights are updated by multiplying the aggregated \(\lambda\)-values with these gradients by using an update function such as gradient descent.

**Listwise Learning to Rank** Two famous types of listwise learning to rank algorithms are *LambdaRank* and *LambdaMart* (Burges, 2010). The latter is a state-of-the-art ranking algorithm and was originally planned to be used at Blendle as well. However, experiments with the implementation of LambdaMart did not yield better results.

Both methods, LambdaRank and LambdaMart, use NDCG as optimization objective. LambdaRank uses the \(\lambda\)’s that were first introduced in RankNet as forces that either push items in a ranking up or down, depending on whether this item was correctly or incorrectly ranked above another item. Moreover, the \(\lambda\)-values are slightly modified in such a way that the difference in NDCG score that is obtained by swapping the two items in the ranking is taken into account as well.

LambdaMart replaces the neural networks with a boosting tree model called MART, described by Friedman (2001). In a boosting tree model a feature vector is sent through a forest of regression trees. Every single tree in the forest yields a certain score. These scores are linearly added to compute the final scores.

Covington et al. (2016) describe the algorithm that is used in the YouTube recommender system. One step in this algorithm is the candidate ranking, for which a deep neural network architecture is used.
3.1.1 Offline learning and online learning

Up until now we have assumed that we know the parameters of a ranking model, for example the weights in Equation 3.1. However, these parameters need to be learned, or at least tuned. This can be done in an offline manner or in an online manner.

In an offline learning setting, the parameters of the model are learned on a pre-made data set. After the learning phase the model is used as it is and the parameters of the model are kept as they are, i.e., they do not update with the behaviour of the users of the system.

In online learning setting on the other hand, the parameters of the model are learned from interaction with the users. Figure 3.1 (Hofmann et al., 2011) summarizes the approach in a ranking setting. The retrieval system constructs a list of items and this list of items is presented to the user. The user then gives feedback to the system. This feedback is mostly implicit and for example measured as a click on an item. (See Section 3.3 for evaluation methods.) This feedback is then used by the retrieval system to update its parameters and generate a new list of items. This circle continues.

Hofmann et al. (2011) describe how to balance exploitation and exploration in online learning to rank. Exploitation uses the parameters learned by the user’s feedback. However, we cannot be sure that we have shown the user everything he or she likes. Therefore, we need to keep exploring the search space. We start with a parameter vector that comes from solely exploiting the information we have about this user so far. Then from this exploiting vector we construct an exploring vector, that is slightly different from our exploiting vector. We use this exploring vector to compute the ranking. If we receive positive feedback from the user, we move our previous exploiting vector in the direction of our exploring vector, if not, we do not change the exploiting vector. We repeat this procedure until convergence.
3.1.2 Comparing rankings

We already briefly mentioned the existence of ranking similarity metrics when we looked into the research on feature selection in Section 2.2. Ranking similarity metrics are, amongst others, also used when evaluating ranking algorithms (e.g. Jiang et al., 2009; Wauthier et al., 2013) and we use them in the current research. There are several metrics that can be used to measure ranking similarity, such as Spearman’s rank correlation coefficient (Spearman, 1904), Kendall’s τ (Kendall, 1938) and the AP Ranking Correlation Coefficient (Yilmaz et al., 2008). Kendall’s τ score is very commonly used in the field of Information Retrieval and is given by

\[ \tau = \frac{C - D}{N(N - 1)/2}, \]  

(3.4)

whereby \( C \) stands for the number of concordant pairs, \( D \) for the number of discordant pairs and \( N \) for the number of items in the ranked list. Concordant pairs are defined as \( x_i > x_j \) and \( y_i > y_j \), where \( j \) follows \( i \), yet \( i \) and \( j \) do not have to be directly adjacent. Discordant pairs are defined as \( x_i > x_j \) and \( y_i < y_j \) or \( x_i < x_j \) and \( y_i > y_j \). The denominator represents the number of pairs in the two ranked lists, as

\[ \binom{N}{2} = \frac{N(N - 1)}{2}. \]  

(3.5)

\( \tau \) ranges from −1 to 1, where a score of 1 means that rankings are identical, whereas a score of −1 means that two rankings are each other’s opposite.

Using the Kendall’s τ metric, differences in all parts of the ranking are given equal importance. Using a similar example as Yilmaz et al. (2008), imagine a default ranking, \( r_1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] \) and two rankings that one wants to compare to the default ranking, \( r_2 = [5, 4, 3, 2, 1, 6, 7, 8, 9, 10] \) and \( r_3 = [1, 2, 3, 4, 5, 10, 9, 8, 7, 6] \). These two rankings, \( r_2 \) and \( r_3 \) receive the same Kendall’s τ score. Yet, it can be argued that items in the top of the ranking are more important to be ranked in the correct order than items in the bottom of the ranking, i.e. \( r_2 \) should get a higher score than \( r_3 \).

Yilmaz et al. (2008) address this issue by inventing the AP Ranking Correlation Coefficient, that is based on both Kendall’s τ and the Average Precision (the area under the precision-recall curve) and only looks at the items above a certain item and not at the items below that item. The score is given by

\[ \tau_{AP} = \frac{2}{N - 1} \sum_{i=2}^{N} \left( \frac{C(i)}{i - 1} \right) - 1, \]  

(3.6)

in which \( C(i) \) is the number of items above item \( i \) that have a higher score than item \( i \) itself (and are thus ranked in the correct order in comparison to item \( i \)). \( N \) is the number of items in the ranking. It can be easily seen that rankings that are ranked in the correct order receive a score of 1 and rankings that are ranked in the opposite order receive a score of −1. −1 is subtracted by the score and the score is multiplied by 2 in order to have the same domain as the Kendall’s τ measure.
3.2 LIME

In chapter 2 we gave a theoretical introduction to LIME (Ribeiro et al., 2016), which we use as one of the baselines for this research. In this chapter we present the implementation details of the LIME pipeline in detail and we describe how we apply LIME to the current research.

3.2.1 LIME pipeline

The LIME pipeline contains two main steps: a training step and an explaining step. In the training step an explainer is built based on training data. In the explaining step this explainer is used to explain the label of a new instance. Note that in what follows we use “data point” for a single instance in the data and we suppose that this data point consists of “feature values”. That is, a single data point is considered to be a one dimensional vector of feature values.

In the training step the LIME algorithm investigates, per feature, the distribution of corresponding feature values in a training data set. In order to do so, LIME makes bins and it divides the feature values over these bins. Like this, the occurrence frequency of each bin is computed. The number of bins depends on parameter choices. In the default setting, which is the setting that we use as well, feature values are divided over quartiles. As these quartiles depend on the data values that have been seen, it is important that a wide variety of data values is used in this training step. E.g. if for a certain feature, values between 0.0 and 1.0 have been observed, these quartiles could become 0.0 − 0.25, 0.25 − 0.5, 0.5 − 0.75 and 0.75 − 1.0, whereas if only a single feature value had been observed for this feature value (for example due to data sparsity) this negatively influences the explaining power of the algorithm. LIME treats continuous feature values different than categorical feature values: in a later stage the newly learned “distributions” (from the division of values over bins) are used to sample from. Categorical features lead to discrete distributions and self-evidently only the discrete values can be sampled and nothing in between those.

Explaining step Now LIME has made its explainer, it can use this explainer to explain new data points. Broadly, to explain a new data point that comes in, this data point is randomly perturbed and in this way neighbourhood data is constructed. These new data points are classified and by doing so an interpretable model is learned locally around the data point that is to be explained. This new interpretable model is used to find which features are most important for the classification score of this data point.

In particular, one sets a parameter for how many neighbouring data points are to be made. The default value is 5000. Now the original data point is discretized. That means, all feature values are divided over one of the, in our case, quartiles that were made for each feature by the explainer. Now we sample from these quartiles, per feature, as many values as number of samples that we want (i.e. 5000 by default). These sampled feature values are rewritten in binary format. If the sampled feature value was equal to the original feature value, the sampled feature value is replaced by 1, if not, the sampled feature value is replaced by 0. The original sampled data (i.e. from the discretized feature values) are kept
as well. Once this is done for all feature values, the newly sampled, discretized feature values, are “undiscretized”. That means, ‘rewritten’ to feature values that could have appeared in the data. The binary data and the undiscretized data are used for the construction of the explanation.

The undiscretized data points are classified by the black box classifier, which classifications we want to explain. For the binary data, the relative distances to the data point to be explained are computed, as data points that are further away from the data point that is to be explained, are less important. The importance of the data points is computed by a kernel function. By default, a Gaussian kernel is used, as stated in chapter 2 and repeated here, given by

$$
\pi_x(z) = \exp\left(-\frac{D(x, z)^2}{\sigma^2}\right).
$$

A regressor is fit on the perturbed data points weighted with $\pi_x(z)$. By default Scikit’s Ridge regressor is used, which performs a linear least squares regression, with L2 regularization. Per feature, the coefficient of that feature in the model is returned. Features with the highest positive coefficients are assumed to be most important to make the model classify the data point in the predicted class, whereas features with the lowest negative coefficients are assumed to work in favour of a different class.

### 3.2.2 Applying LIME to the current research

We choose to use LIME as our baseline for two reasons. First, LIME was a new state-of-the-art approach at the time of conducting this research and secondly, LIME is designed to be model-agnostic, a characteristic that we also aim for in our listwise model. As mentioned, we are bound to the Blendle log-linear ranking scoring function. Yet, we also want our approach to work on more state-of-the-art ranking functions such as LambdaMart. In this section we briefly describe how we apply LIME in the current study.

**From ranking function to classifier** LIME is designed to explain the explanations of any classifier. However, we deal with a ranking function. Therefore we need to think of our ranking function as a classifier. In order to do so, we bin our ranking scores. We use the smallest ranking score in our training data as the start of our range, the largest ranking score in our training data as the end of our range. (For a precise description of the training data, and the data in general, see chapter 7.) Scores that are out of this range are placed in the first bin if they are smaller than the scores that are placed in the first bin and in the last bin if they are larger than the scores that are put in the last bin. In order to clearly distinguish between the original version of LIME and our modified version of LIME we will use the term mLIME when we specifically refer to this modified version of LIME.

**Dealing with similar scores** In the current data set users have the same scores for many of the feature values. E.g., we use a feature that captures negative feedback, yet many users do not give negative feedback. Therefore, certain configurations of feature values occur more often than other configurations.
This causes many scores to be binned in the same value range in the training step of LIME. As a consequence, if LIME has to sample from its constructed distributions, the possible sample values do not vary a lot. As the sampling is a non-deterministic process and the differences between the feature values are so small, the features that are found to be the most prominent in the regression vary. This is a drawback of the LIME algorithm and something that we solve when constructing our own listwise explanation method.

3.3 Evaluation in Information Retrieval

In this section we describe several evaluation techniques that are used in the field of Information Retrieval. At the end of this section we look into the evaluation of explanation systems.

Implicit and explicit feedback  A commonly made assumption within the field of Information Retrieval is that a click on an item represents the fact that a user is satisfied with this item. This is a form of implicit feedback. That is, we cannot be fully certain that a click on an item means positive feedback and we can definitely not assume that no click on an item means negative feedback. At most a “non-click” means as much as ‘another returned item that was clicked on was probably a better fit for this particular user’. Even though implicit feedback is less reliable than its counterpart, explicit feedback, this form of feedback is often preferred as evaluation method over explicit feedback. Users do not tend to give explicit feedback, whereas implicit feedback is given every single time a user uses the system.

A/B testing  A method that is often used to find out whether a newly implemented method works better than the previous state-of-the-art is A/B testing. In an A/B-test users are randomly divided over two groups; group A and group B. One of the groups is shown the new implementation, whereas the other group is shown the old implementation. If the new implementation scores significantly higher on the evaluation objective (for example number of clicks) this implementation can be used for all users.

There are several ways to divide users over groups. The preferred way depends on the situation that is tested. Imagine a front-end test in which the tester wants to find out whether a certain homepage design improves the conversion rate, i.e. whether more users sign up for the service. In this case, it is arguable to flip a coin every single time a new user arrives at the website. The coin flip solely decides in which group a user is classified.

Different situations require a different approach. Sometimes one knows in advance which users are to be divided over groups. This is the case in the current research, namely, all Blendle users. Again, one can use a random approach to divide the users over groups. A drawback of a pure random approach is that one does not take the difference between heavy and non-heavy users into account. There may be a few heavy users, that, if they all end up in the same group by coincidence, skew the equal behaviour of the groups that one strives for. One way to solve this issue is by using stratified sampling, for example described by Deng et al.
Table 3.1: Examples of outcomes of rankings composed by Balanced interleaving and by Team-Draft interleaving, by Radlinski et al. (2008).

<table>
<thead>
<tr>
<th>Rank</th>
<th>Input Ranking</th>
<th>Interleaved Rankings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>c</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td>5</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>6</td>
<td>f</td>
<td>g</td>
</tr>
</tbody>
</table>

(2013). We explain this concept with the current application in mind. In order to prevent all heavy users ending up in the same group, we can sort the users based on their historic reading behaviour. We can divide this ranking in so-called strata. These strata contain groups of users that approximately read the same number of articles in the period that we used to sort them. Now if we randomly divide the users in a stratum over the groups in the A/B-test and we do this for all strata, we ensure that both heavy and non-heavy users are represented in each group of the test.

Moreover, one needs to make sure that one excludes bots from the test. Web pages are often visited by bots that crawl over the entire website and again skew the, in our case, reading counts of the users in the different groups.

In the current research we use A/B testing, with stratified sampling and bots excluded from our experimental design.

Interleaving  Whereas during an A/B-test the different implementations are kept separately, the implementations are mixed when using interleaving as evaluation method. Therefore, interleaving is especially suited for the evaluation of ranked systems. We describe several interleaving methods below.

The ranking that is shown to the user is composed by consecutively adding items from the exploitative ranking and from the exploratory ranking. In Balanced interleaving (Radlinski et al., 2008) one chooses a ranking to start with, adds its first element to the new ranking and then adds the first element of the other ranking, unless this element was already added. In that case the next item from the start ranking is chosen (unless this was also already added, etc.). One continues until there are no items left in any of the rankings. On the other hand, Team-Draft interleaving (Radlinski et al., 2008) is based on selecting players for a team in a non-professional setting. There are two team captains, captain A and captain B. The team captains can alternately choose a player, i.e. an item to add to the list from their own selection. The captain that can choose first is decided with a coin flip. Every single time a captain chooses, he is ought to choose the best player of his preferred team, i.e. the item that is placed highest in his list. Table 3.1, by Radlinski et al. (2008), shows the outcome of both interleaving methods.
Statistical evaluation of online tests  There are several ways to evaluate the results of an online test. In the current research we use an A/B-test (as described above) to answer parts of our research questions and, amongst others, we use a randomization test to evaluate this test. Therefore we briefly elaborate on this specific test here. Again, we explain the concept with the current application in mind. One randomly divides users over groups. One does this \( N \) times. This way one can make a probability distribution that expresses how likely it is that users in a group behave a certain way if they were randomly assigned to this group. Now one can use this distribution to find out whether an observed effect is likely to happen by chance or not. That is, one states:

\[
H_0 \; - \; \text{The effect is likely to happen by chance.}
\]

\[
H_1 \; - \; \text{The effect is not likely to happen by chance and is caused by the treatment.}
\]

If the observed effect occurs in less than \( \alpha \% \) of the times in the random division, one can reject the \( H_0 \). In this study we use \( \alpha = 5\% \) as significance level.

Evaluation of explanation methods  As mentioned, the evaluation of explanation methods is not straightforward, as often one does not really know why a system behaves the way it does. Therefore, it is not easy to make labeled data. Often, explanation systems are evaluated by asking users of the system whether they are satisfied with a given explanation (e.g. Bilgic and Mooney, 2005; Herlocker et al., 2000; Ribeiro et al., 2016). Several studies have looked into some sort of offline evaluation. For example, Ribeiro et al. (2016) evaluate the faithfulness of LIME on interpretable classifiers. The authors make sure these classifiers only use a predefined number of features for the classification. These features are called the “golden features”. Then they generate explanations for predictions on a new data set and measure how many of the golden features are recovered. LIME scores high on this approach, although it can be questioned whether this evaluation metric really measures the faithfulness it is aiming for.

3.4 Neural Networks

In our study we use neural networks for efficiency reasons. We train an end-to-end model that we use to make explanations very quickly and thereby bypassing computationally expensive models. Therefore, we briefly look into the theoretical background of neural networks in this section. Readers that are experienced in the field may want to skip this section. Neural networks have a long history and have known ups and downs in their popularity. At the time of writing they are unprecedentedly popular, as is the entire field of Machine Learning.

The goal of a neural network is to approximate a function that maps input data to output data. Most networks consist of multiple linear transformations, that are all followed by a non-linear function. Consecutive computations are often called layers and layers are said to consist of nodes, which are the values in the vectors that traverse through the network. The structure of such a network can be given by

\[
a_L(x; \theta_1, \ldots, \theta_L) = h_L(h_{L-1}(\ldots h_1(x, \theta_1), \theta_{L-1}), \theta_L),
\]
whereby $x$ is the input to the network, $\theta_l$ are the parameters for layer $l$ and $a_l = h_l(x, \theta_l)$ is a (non-)linear activation function.

The parameters of the network, $\theta_l$ are learned from the data and optimized such that a loss function is minimized. This loss function is minimized by feeding input data to the network and computing the loss on the output data. This is called the forward pass of the network. The parameters of the network are then slightly changed with the aim to decrease the loss. This updating function is often given by

$$\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla_{\theta} L,$$

in which $\eta$ is a learning rate, that can be used to increase or decrease the parameter update value and $L$ is the loss function. There are several variations to this parameter update, which we discuss below. This process continues for a number of iterations, until the loss value does not decrease anymore. Below, we describe the different steps in training a neural network in more detail.

### 3.4.1 Weight initialization

The parameters of a network are often called the *weights* of the network. The initialization of these weights has proven to influence the final power of the network. Clever weight initialization can improve the quality of the network. In general we need to find the correct trade-off between initializing the network with small weights and initializing the network with large weights. If the weights are too small, at some point the updating signal is too weak to learn from. On the other hand, if the weights are too large, the opposite happens. In this section we discuss several weight initialization techniques.

**Random or zero initialization** A naive approach is to initialize the weights with random values (perhaps values in a certain range), or with zeros. The latter approach, zero initialization, is in general a bad idea. One should aim to initialize the weights to be asymmetric, as otherwise all nodes in the network will have the same gradient. This prevents the network from learning.

**Xavier initialization** A method that has proven to work very well is *Xavier*-initialization (Glorot and Bengio, 2010). Here the weights are initialized by randomly sampling values from a normal distribution given by

$$\theta = \mathcal{N}(0, \frac{2}{n_i, n_{i+1}}),$$

where $n_i$ are the number of nodes in the current layer and $n_{i+1}$ are the number of nodes in the next layer. This is the method that we use in the current research (see chapter 6 and chapter 7).
3.4.2 Activation functions

One could choose from multiple activation functions to capture the non-linearities in the data. In this section we describe a number of possible activation functions.

**Sigmoid like activation functions**  Non-linearities as the sigmoid function, given by

\[
\sigma(x) = \frac{1}{1 + e^{-x}},
\]  \hspace{1cm} (3.11)

or the tanh-activation function, given by

\[
tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}},
\]  \hspace{1cm} (3.12)

have been popular due to their intuitive domain. That is, the domain of the sigmoid is \([0, 1]\) and the domain of the tanh-function is \([-1, 1]\). This intuitively reflects whether a node in the network is activated or not.

**ReLU and Leaky ReLU**  More popular these days is the ReLU activation function (Krizhevsky et al., 2012). ReLU stands for Rectified Linear Unit. The ReLU function is given by

\[
ReLU(x) \begin{cases} 
  x, & \text{if } x > 0 \\
  0, & \text{otherwise.}
\end{cases}
\]  \hspace{1cm} (3.13)

Inspired on the ReLU activation function is the leaky ReLU, given by

\[
Leaky ReLU(x) \begin{cases} 
  x, & \text{if } x > 0 \\
  ax, & \text{otherwise.}
\end{cases}
\]  \hspace{1cm} (3.14)

In this research we use the ReLU activation function, again see chapter 6 and chapter 7.

3.4.3 Weight optimization

There are several ways to update the weights. In this section we discuss some well-known methods.

**Stochastic Gradient Descent**  The update function for gradient descent is given in equation 3.9. We can choose to update the parameters based on one data point, based on all data points, or based on parts of the data points. Normal gradient descent computes the updates based on all data points. This has several disadvantages. First, the gradients that are computed may optimally fit the training data, but may not optimally fit the data in general. Secondly, it is very time consuming. Therefore, stochastic gradient descent is often used. A number of data points are chosen stochastically from the entire data set and the updates
are computed based on these data points. In the next iteration, new data points are stochastically chosen, etc.

**Adagrad** The Adagrad update rule (Duchi et al., 2011) is similar to stochastic gradient descent, that was described above, yet, the learning rate is adaptive. Large updates are assigned to frequent parameters, whereas small updates are assigned to infrequent parameters. The update rule is given by

\[ \theta(t+1) = \theta(t) - \frac{\eta}{\sqrt{m} + \epsilon} \nabla_{\theta} L, \]  

whereby \( \epsilon \) is a small value to avoid zero divisions and \( m \) is given by

\[ m = \sum_{\tau} (\nabla_{\theta} L)^2, \]  

where \( \tau \) describes the sum up until time stamp \( \tau \).

Other examples of update rules that make use of an adaptive learning rate are *AdaDelta* (Zeiler, 2012), *RMSprop* (Tieleman and Hinton, 2012) and *Adam* (Kingma and Ba, 2014).

**Adam** Adam (Kingma and Ba, 2014) also uses a stochastic gradient descent based updating function, with an adaptive learning rate, yet it also uses the idea of *momentum* (Sutskever et al., 2013). The intuition behind momentum is that gradients also keep some the direction, i.e. the *momentum*, of the previous gradients. The update rules for Adam are given by

\[ \theta(t+1) = \theta(t) - \frac{\eta}{\sqrt{v_t} + \epsilon} \hat{m}_t, \]  

whereby \( \hat{m}_t \) and \( \hat{v}_t \) are given by

\[ \hat{m}_t = \frac{m_t}{1 - \beta_1^t} \]  

and

\[ \hat{v}_t = \frac{v_t}{1 - \beta_2^t}, \]

where \( \beta_1^t \) and \( \beta_2^t \) are values close to 1 and \( m_t \) and \( v_t \) are given by

\[ m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{\theta} L \]  

and

\[ v_t = \beta_2 v_{t-1} + (1 - \beta_2) \nabla_{\theta} L^2. \]

In the current research we use Adam as weight optimization technique. Again, the reader is referred to chapter 6 and chapter 7 for precise implementation details of the current study.
3.4.4 Backpropagation

In the previous section we have looked into several ways to optimize the weights. However, we have not described how to compute $\nabla_{\theta} L$. In this section we look into this computation.

After the forward pass, the loss (the error) is computed for that iteration. This error is propagated back through the network. This is done with an algorithm called backpropagation (Rumelhart et al., 1988).

If one derives the backpropagation algorithm, one finds that updating the parameters of a layer in the network can be done by computing so called $\delta$’s for each layer. These $\delta$’s are computed by taking the derivative of the loss function with respect to the input parameters of a layer. Finally the derivative of the loss function with respect to the weights can be shown to be

$$\frac{\delta L}{\delta \Theta_l} = \delta_l z_{l-1}, \quad (3.22)$$

whereby $\Theta_l$ are the parameters at layer $l$ and $z_{l-1}$ is output of the previous layer.

3.4.5 Overfitting

Overfitting is the problem of learning the patterns in the training data “too well”; so well, that they cannot be extended to new data that the network has not seen before. There are many solutions to avoid overfitting. We describe a few of them in this section.

**Training, validation and test set and early stopping**  It is important to divide the data in a training set, a validation set and a test set. The training set is used to train the network, the validation set is used to train the hyper parameters and the test set is strictly kept apart and only fed to the network at the end, to test whether the network also performs well on data it has not seen before. Early stopping refers to the fact that one should not minimize the loss function on the training data as much as possible, but stop in time, so that the learned patterns can still be generalized to other parts of the data set. Figure 3.2 shows this graphically.
Weight regularization  Another common way to avoid overfitting is weight regularization. By adding a regularization term to the loss function, one avoids weights becoming too large. A very common way of weight regularization is $L_2$-regularization. This adds

$$\frac{\lambda}{2} \sum_i ||\theta_i||^2$$  \hspace{1cm} (3.23)

to the original loss function. $\lambda$ is a hyper parameter and is often chosen to be something like $10^{-1}$ or $10^{-2}$.

**Dropout** Overfitting can also be avoided by using dropout (Srivastava et al., 2014). In this approach nodes in the network are randomly dropped, with a certain probability. This forces nodes to be able to work together with all kinds of nodes, as it can happen that its direct neighbour is dropped out.

In the current research we implement all of the above techniques to avoid overfitting.

### 3.4.6 Batch normalization

A last good practice that is worth mentioning and that we use in the current research is batch normalization (Ioffe and Szegedy, 2015). After applying back-propagation, the input distributions of the layers have changed. Therefore, we normalize the inputs again, so that we use input from the same distribution at all times.

**Evaluation** There are multiple ways to evaluate a neural network. We minimize a mean-squared error loss given by

$$\mathcal{L} = \frac{1}{N} \sum_i (\hat{y}_i - y_i)^2.$$ \hspace{1cm} (3.24)

We compute the accuracy as the ratio of number of reasons that are correctly found. I.e. we use precision at $N$, whereby $N = 3$ in our case, the number of reasons we want to be able to return.
Chapter 4

Problem Setting

We conduct this research in the context of Blendle and therefore we define the precise problem setting in this chapter. Note that even though we use this specific problem setting, we design a method that could be used to explain the rankings of any ranking algorithm, not only the ranking of the Blendle recommender system.

4.1 The Blendle pipeline and vocabulary

In this section we look into parts of the Blendle pipeline in more detail. Moreover, we need to introduce some ‘Blendle vocabulary’ that will be used throughout this work.

During the day Blendle’s editorial staff selects news items from newspapers and magazines. A so-called pick predictor helps them with their work. This pick predictor predicts how likely it is that a particular item will be selected, i.e. ‘picked’ by the editorial staff. This helps them to focus their efforts on the most promising items. This pick predictor is developed and trained by Blendle and uses historical picks as its training data.

Items are from a certain provider and of a certain channel or a number of channels. The provider refers to the newspaper or magazine and a channel refers to a particular topic. For example, an item can have the channels “foreign affairs” and “opinion” assigned to it.

Figure 4.1 shows the part of the pipeline that is used to score and rank documents. Once the news items are selected by the editorial staff, they enter the scoring and ranking pipeline. Each document that enters the pipeline is first preprocessed. During this preprocessing natural language processing steps are applied, such as tokenization, topic selection, entity enrichment, etc.

Features The features that are used to give the article its score are based on these enriched documents and the user for whom the article is scored. The features are manually designed. All features that are used at the time of writing are summarized in Table 4.1. Features are not static over time. E.g., how much feedback a user has given on a certain provider, or whether a user tends to buy
from a specific channel is subject to change. Moreover, when new features are introduced, we cannot always rely on historical data yet. For example, when the negative feedback feature was introduced, there was no negative feedback data for users available yet, as before the introduction of the feature users were not able to give negative feedback. Therefore the scores for a feature can be very similar for all users when a feature has just been introduced. Especially for this reason, it is recommended to retrain an explanation model that relies on these features not only after introducing new features, but also once in a while in between the introduction of new features. The model is less sensitive to changing reading behaviour of individual users, as it is trained on many users and a changed individual user is often comparable to a user that was encountered during the training of the recommendation model.

**Ranker** Based on the feature values, ranking scores are computed by a log-linear ranker. This log-linear ranker is given by

\[ s(w, f) = \sum_i w_i \log(f_i), \]  

in which \( f_i \) are the feature values that were described in the previous section and \( w_i \) are the weights of these feature values. These are computed in an online fashion (see Section 3.1.1). Users only receive a personalized ranking if they have read more than a certain number of articles. Until they reach that number, users receive a standard selection of articles.
Table 4.1: Features that are used in the Blendle pipeline and their descriptions.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item rating score</td>
<td>The rating that the item has received by the editorial staff</td>
</tr>
<tr>
<td>item pick probability</td>
<td>How likely it is that this item gets picked by the editorial staff (recall the pick predictor that was described in Chapter 2)</td>
</tr>
<tr>
<td>item number of images</td>
<td>The number of images that this item contains</td>
</tr>
<tr>
<td>item channel followed by user</td>
<td>The user follows (some of) the channels of the item</td>
</tr>
<tr>
<td>item provider followed by user</td>
<td>The user follows the provider of the item</td>
</tr>
<tr>
<td>newsletter purchased channel score</td>
<td>How much a user tends to buy from this channel given his bundle</td>
</tr>
<tr>
<td>newsletter purchased provider score</td>
<td>How much a user tends to buy from this provider given his bundle</td>
</tr>
<tr>
<td>user item negative channel feedback</td>
<td>How much negative feedback a user has given on this channel</td>
</tr>
<tr>
<td>user item negative provider feedback</td>
<td>How much negative feedback a user has given on this provider</td>
</tr>
</tbody>
</table>
The ranking scores are stored for later use. Each morning, all documents and their scores are retrieved for each user. Based on these scores, the personalized ranking for each user is constructed.

**Diversification** Because Blendle users prefer diverse bundles over bundles with articles that suit them very well, yet are all very similar (for example because they are all from the same newspaper), the user’s personalized bundle is diversified after the retrieval of the computed scores has finished. For the diversification the Maximal Marginal Relevance (MMR) algorithm (Goldstein et al., 2000) is used. MMR is originally proposed for the task of document summarization and aims to capture a metric that combines the relevance of a sentence in a document and the similarity of this sentence to the sentences that were already extracted. In the current context the MMR score is used to find the correct balance between relevance of a new item in a user’s selection on the one hand, and the similarity with the items that are already selected on the other hand.

The formula for MMR is given by

\[ \lambda \text{Rel}(i) - (1 - \lambda) \max_{s \in S} \text{Sim}(s, i), \]  

where \( \lambda \) is a hyper parameter that defines how much focus is assigned to each of the two components, \( i \) is the item, \( S \) is the user’s selection so far and \( s \) is an item in this selection. At Blendle, \( \lambda \) is currently set to 0.3. The relevance score of the item is the ranking score. The first item in the ranking is simply the item with the highest ranking score. After that, each new item is the item that has the highest MMR. I.e. the MMR score is computed for all possible items in a users ranking.

**Blendle for the users** Users receive their selection on blendle.com, in the Android app and/or iOS app. Figure 4.2 gives an impression of what a part of a personal selection could look like. Moreover, the editorial staff writes a newsletter that contains the personal selections for the user together with a written recommendation that gives a very brief summary of the article. These newsletters are sent by e-mail to the users that have opted in.

**4.2 Explanations for the Blendle recommender system**

In this research we make explanations for the ranking that is produced by the Blendle recommender system, of which the pipeline was described in the previous section. That means that we add an additional module to Figure 4.1 behind the final ranking, that takes this ranking as its input and produces explanations as its output. In order to do so, we first need to answer our third research question, that investigates how we can provide users with easy to understand, uncluttered, listwise explanations. In Section 6.1 we answer this research question. Blendle already uses some heuristic justifications as reasons (recall the distinction between justifications and descriptions by Vig et al. (2009) that we described in
Figure 4.2: Impression of a personal selection on blendle.com. Users are presented with article cards. Users can read the article by clicking this card.

Table 4.2: Heuristic Blendle explanations.

<table>
<thead>
<tr>
<th>Reason</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Because you often read about CHANNEL.</td>
</tr>
<tr>
<td>1</td>
<td>Because often read from PROVIDER.</td>
</tr>
<tr>
<td>2</td>
<td>Because you are interested in CHANNEL.</td>
</tr>
<tr>
<td>3</td>
<td>Because we think CHANNEL could be interesting for you.</td>
</tr>
<tr>
<td>4</td>
<td>Because we think PROVIDER could be interesting for you.</td>
</tr>
<tr>
<td>5</td>
<td>Because you follow CHANNEL.</td>
</tr>
<tr>
<td>6</td>
<td>Because you follow PROVIDER.</td>
</tr>
<tr>
<td>7</td>
<td>Because you often read from PROVIDER.</td>
</tr>
<tr>
<td>8</td>
<td>Because you seem to like a long read every now and then.</td>
</tr>
<tr>
<td>9</td>
<td>Because you often read from PROVIDER.</td>
</tr>
<tr>
<td>10</td>
<td>The editors really liked this piece.</td>
</tr>
<tr>
<td>11</td>
<td>According to the editors, this is one of the best stories of the day. No matter your preferences.</td>
</tr>
</tbody>
</table>

Section 2.1). We use these as one of the baselines of our research. Properties of the articles and the user are compared and from this comparison reasons are constructed. An example of this could be a long article that is recommended to a user. This particular user may tend to read long articles in general and therefore a justification could be something along the lines of “because you seem to like longer articles”. Of course, this is not necessarily the real descriptive explanation for why a user sees this article. That means this approach is not faithful. Our new listwise approach solves this issue. All heuristic reasons are summarized in Table 4.2. To show the users a wide variety of reasons, the reasons that are presented to the user are diversified in way that is comparable to the method that was used for the item diversification.

We use the Blendle environment to answer our research questions. We replace the heuristic Blendle reasons with faithful listwise explanations that can be produced in real time. We test our algorithm in an offline and in an online manner. For
the online evaluation we conduct an A/B-test on all Blendle users who receive a personalized selection of news articles, which is a unique opportunity of the current setup — we are not dependent on the regular data sets such as the MovieLens dataset or the Netflix dataset. With the results of the offline and online evaluation we can answer our last research question, namely whether our listwise explainer performs better than our two baselines on explaining ranked items to users.

Before we answer these three research questions, we conduct a user study to validate our assumption that users would like to receive explanations for their ranking. We also investigate how users would like to receive these explanations. We answer these two research questions in the next chapter.
Chapter 5

Do news consumers want explanations for their personalized news rankings?

In Chapter 1 we have introduced two main reasons why it is important to explain algorithmic decisions: the users of an algorithm are expected to gain more trust in the decision of the algorithm and are better able to give feedback if needed. Moreover, explanations help the developers of an algorithm to spot potential mistakes. Up until now we have silently assumed that users want explanations of algorithmic decisions. Yet, it is important to validate this assumption, especially because adding explanations is for a large part done for the benefit of the user. Research by for example Herlocker et al. (2000) and Tintarev (2007) point in the direction that users would like to receive explanations, but we cannot simply assume that this holds for all applications and therefore also for the current application. Moreover, the recommender system research and the Artificial Intelligence research in general have not been static since the time these studies were conducted and therefore the user experience may have changed. In order to find out whether users would find it helpful to receive explanations, we conduct a user study amongst Blendle users in which we ask exactly this question. We also investigate what explanations should look like and what types of design users prefer. I.e., in this chapter we aim to answer our first two research questions.

This chapter is based on ter Hoeve et al. (2017).

5.1 Experimental setup

We sent out an email questionnaire to a selection of Blendle users, 541 in total. Approximately two third of these users had a Blendle Premium subscription at the time of sending. The rest of these users used the micropayment system, but had used Blendle Premium at least once, for example via a free trial that lasted for one week.

We designed five different types of reasons to explain our recommendations, to
be judged by participants in the study. Table 5.1 summarizes all five reason types. Visible reasons are reasons that can be found on the card (e.g., the topic or the length of the article), invisible reasons are reasons that cannot be found on the card itself (e.g., the author). Figure 5.1 shows examples of items that were shown to participants.

Participants were shown three different reason types and subsequently asked to answer five questions per type. To limit the length of the survey, participants were asked to judge three types of explanations, out of the five described above. Figure 5.2 shows an example of the interface of the questionnaire. To make sure the results are not biased by the type or content of an article, three different articles were considered: 179 users were sent the first article, 180 users were sent the second, and 182 users were sent the third article.

Note that users were not sent the entire article, but only the introduction card to the article. This article card contains a picture, a brief introduction to the article, the name of the newspaper or the magazine, a topic, the approximate reading time of the article, how many people liked the article and the reason type. The card functions to give the news consumer a brief introduction to the article to decide whether he or she would like to read it. Figures 5.1a, 5.1b, 5.1c show the three different types of article cards that are used. Note that users are randomly divided over all three article types and over reason types. That is, no personalization was used here. We did not, however, completely randomize the order in which participants answer questions. First, users are either shown reason type 1 or 2, then 3 or 4. All users are shown reason type 5, as reason type 5 is very different from the other reason types. In three final questions participants are asked to fill in their age and gender and whether they would like to add some final remarks (if any).

The questions that were asked for each participant are detailed in Table 5.2. First, we ask participants whether they would find explanations useful and we ask them to choose between yes, somewhat, no or I don’t know as possible answers. We then show several examples of explanations and ask participants to judge the examples on four Tintarev (2007)’s dimensions: transparency, sufficiency, trust and satisfaction, all on a five point scale. We decided to omit questions on Tintarev (2007)’s scrutability, efficiency and effectiveness as metrics, as participants are not confronted with their own personal selection of news. For this reason, they will not be able to reliably judge whether they would open this article. Note that if participants have selected no or I don’t know as a reply to whether they would like explanations, we tell them we would still like to show them some possible ways of explaining their articles and ask for their judgment.

### Table 5.1: Reason types used in the user study.

<table>
<thead>
<tr>
<th>Reason type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Single reason, visible</td>
<td>Because you like politics</td>
</tr>
<tr>
<td>2. Single reason, invisible</td>
<td>Because you like this author</td>
</tr>
<tr>
<td>3. Multiple reasons, visible</td>
<td>Because you like politics and long articles</td>
</tr>
<tr>
<td>4. Multiple reasons, combined</td>
<td>Because you like <em>De Tijd</em> and this author</td>
</tr>
<tr>
<td>5. Bar chart</td>
<td>See Figure 5.1e</td>
</tr>
</tbody>
</table>
(a) Single reason, visible – “Because you like reading about politics.”

(b) Single reason, invisible – “Because you like long reads and tech.”

(c) Multiple reasons, visible – “Because you often read from this author.”

(d) Multiple reasons, combined – “Because you follow De Tijd and read from these authors more often.”

(e) Bar chart – “Selected for you based on:
  Author(s): Maarten Keulemans;
  Publication: De Volkskrant;
  Topic: Tech”

Figure 5.1: Examples of reason types as shown to users in our user study. Textual reasons are in the lines that start with “Omdat” (because). For the bar chart layout the reasons starts with “Voor jou gekozen” (selected for you). Translations are given below each article.
Figure 5.2: Example interface of the questionnaire, for a single question. Judgment at the top (Q4, see Table 5.2).
Table 5.2: Questions used in the questionnaire as part of our user study.

<table>
<thead>
<tr>
<th>Type</th>
<th>Question asked (English translations of the Dutch questions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1. Wants reasons?</td>
<td>On the figure below you can see what an article currently looks like on Blendle Premium. The articles that you see are chosen based on your personal preferences and what you like to read. Imagine we would give you more information about why we chose a certain article for you. Would you find that useful?</td>
</tr>
<tr>
<td>Q2. Transparency</td>
<td>I understand the way that is used to explain why I see this article.</td>
</tr>
<tr>
<td>Q3. Sufficiency</td>
<td>I get enough information to decide whether I would like to read this article.</td>
</tr>
<tr>
<td>Q4. Trust</td>
<td>The reason why I see this article, makes me trust the algorithm that selected this article for me.</td>
</tr>
<tr>
<td>Q5. Satisfaction</td>
<td>I am satisfied with the way in which this article is shown to me.</td>
</tr>
</tbody>
</table>

5.2 Answering research questions 1 and 2

In this section we analyse the results of our user study and answer our first two research questions. The first question is answered in Section 5.2.1 and the second question is answered in Section 5.2.2.

A total of 120 users filled out our survey, of which 41 answered questions about the first article type, 36 about the second and 43 about the third article type. Of these 120 users, 103 users had a Blendle Premium subscription, while 17 users used the micropayment system at the time of sending out the survey. As there are not enough responses of non-premium users to put them in a separate group, we perform our analysis on all respondents together.

5.2.1 RQ 1 - Do users want recommendation reasons?

Table 5.3 shows the results of what users answered to the question whether they would like to see better explained why they see articles in their selection. The significant majority answered yes or somewhat to this question, if compared to the number of people that answered no or I don’t know ($\chi^2 = 14.55, p < 0.001$).

5.2.2 RQ 2 - Do users want a particular type of recommendation reasons?

Table A.1, in the appendix, shows the total average and standard deviation on all three articles combined, as well as the mean and standard deviation per question per article. Table A.2 shows whether the differences in scores for the different types of questions are statistically significant or not. As the answers are
### Table 5.3: Participant answers to Q1: Would you like to see more information on why articles are selected for you?

<table>
<thead>
<tr>
<th>User wants reasons</th>
<th>Times answered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>65</td>
</tr>
<tr>
<td>Somewhat</td>
<td>24</td>
</tr>
<tr>
<td>No</td>
<td>26</td>
</tr>
<tr>
<td>I don’t know</td>
<td>5</td>
</tr>
</tbody>
</table>

Independent, yet not necessarily sampled from the normal distribution, we use the two-sided Mann-Whitney U test, with $\alpha = 0.05$ as significance level. The sample sizes can be found in Table A.3. From these results a few points stand out. First of all, although users do want more information about why they see a certain article, the results do not show a clear preference as to which type of explanation users prefer. Only a few differences were significant (shown in boldface in Table A.2). However, when we correct for the number of comparisons that we make, and take $\alpha = 0.001$ as significance level (using the Bonferroni correction and dividing our original $\alpha$ by 50, the number of comparisons that we make), none of the reason types scores significantly higher than another reason type. Another interesting point to make is that the standard deviations of the scores on the fifth reasoning type are, on average, bigger than the standard deviations of the scores on the other reasoning types, i.e., users either seem to like this way of showing reasons, or they do not.

### 5.3 Implications

In this user study we investigated whether news consumers would like to receive explanations about why articles were selected for their personalized selections of news articles. We also investigated how they would prefer to receive these explanations.

Our questionnaire showed that a large majority of the respondents would like to receive these explanations, yet they do not show a clear preference as to how they would like to see these. More broadly, our research shows that users nowadays attach importance to explanations of algorithmic decisions broader than the domain described in Herlocker et al. (2000).

The user study validates our assumption that users of news recommender systems would like to receive explanations for the articles that are selected for them and it gives us ground to develop such an explanation system in the current research.
Chapter 6

Method to explain rankings: 
(Q-)LISTEN

In the previous chapter we have seen that users of a news recommender system would like to receive explanations for the items they see in their personal selection. In this work we aim to build a system that is able to explain any type of ranking algorithm. In Chapter 1 we have motivated why a listwise method is preferred over a pointwise method. In order to design such a listwise method we first need to answer our third research question, namely, how can we provide users with easy to understand, uncluttered, listwise explanations? In this chapter we first elaborate on this question. Once we have the answer to this question, we present LISTEN, our approach to generate faithful, model-agnostic, LISTwise Explanations. LISTEN needs to run in production, in real time. As LISTEN is not scalable to this extent, we extend it to Q-LISTEN, in which Q stands for Quick. The method uses a neural network to simulate LISTEN’s explanation function. This extremely speeds up the computation time and makes it feasible to run the method in real time. This is one of the important contributions of this research. As neither LIME or mLIME are scalable to run in production either, we also train a neural network to learn mLIME’s explanation function. We will refer to this method as Q-mLIME. We conclude this chapter with a brief overview of how we communicate the explanations to the users.

6.1 RQ 3 - How do we provide users with understandable, uncluttered listwise explanations?

Recall from Chapter 1 that we define listwise explanations as explanations that take the entire ranking into account. That is, items are not explained based on their individual score, but based on their position in the list and their position in comparison to other items in the list. The question how to explain a list is not straightforward to answer. In Chapter 2 we have seen some studies that touch upon this question. For example, Hernando et al. (2013) try to map the entire recommendation space in a graph structure. We have already questioned
whether this is an intuitive way for most users to understand why items are recommended to them.

In Chapter 2 we introduced the notion of a contrastive explanation (Miller, 2017). Intuitively, the contrastive explanation fits very well to the question of how to explain a ranking, as the questions that we want to answer are questions along the lines of “Why is item A positioned at position X in the ranking as opposed to any other item?” Muhammad et al. (2015) construct a system for hotel recommendations and explain these recommendations by taking other hotels into account as well. They construct bar chart contrastive explanations of the form “this hotel has feature X and this makes it better/worse than Y% of the alternatives”. Now, for hotel recommendations this can work, as users have as goal to find a single hotel. For news recommendations however, contrastive explanations are not as intuitively, as in the end we aim to select a given number of “perfect” articles for the user. Comparing the articles with each other would be difficult as then the user could wonder why an article that is worse than Y% of the other articles is even in his or her selection. Comparing the articles with articles that are not in the selection is also somewhat confusing for the user.

Our user study (Chapter 5) showed that users do not have a specific preference as to how recommendation reasons are shown. Yet, a somewhat less “natural” explanation style as the bar chart, was either loved or hated. Therefore we aim to develop an explanation style that is as simple as possible, yet a faithful description of the underlying algorithm.

Taken the above into consideration, we come up with a listwise explanation style that is on first sight indistinguishable from a pointwise explanation style. Yet, the explanations are constructed by taking the entire ranking into account, and not only the score of an individual item. That is, the explanations faithfully describe which features were most important for an item to end at that particular position in the ranking. Recalling the example that was given in Chapter 1, that used equation 1.1 as a simple scoring function, a pointwise explanation for item $d_0$ would be “because you like feature $x_2$”, whereas a listwise explanation would be “because you like feature $x_1$”. A user will not notice the difference in explanation style, but the listwise explanations are more faithful to the model.

As stated in the introduction of this chapter, we call our method LISTEN.

Another decision that we make is to only report these features to the user that actually increase the score of the item in the ranking. We do this in order to avoid explanations such as “you see this article because you do not really like X”, whereas the article is actually about topic X. Even though these could be faithful explanations, they are unintuitive for users of the system. Developers of the system could be helped with explanations in the form of features that decrease the score of an article.

The reasons that we compute, both with mLIME and with LISTEN, are not very understandable for users. We cannot present users with a matrix of importance scores per feature value or with the names of the most important features. Therefore, we rewrite the outcome of the explanation system to sentences that are presentable to users. The goal of this research is not to do this last step automatically and therefore we use a manual constructed one-to-one mapping of features and explanations to users (see Section 6.5).
6.2 LISTEN: a LISTwise ExplaiNer

In this section we describe how we design LISTEN. We use the following intuition: if changing the value of a feature causes the ranking to change a lot, then this feature was important. If the ranking does not or barely change by changing the feature value, then this feature was not important. Algorithm 1 summarizes the general pipeline of our approach. In the remainder of this section we describe the individual steps of the pipeline in full detail.

Algorithm 1 Overview LISTEN
1: Training phase
2: Find the importance of individual feature values by changing these feature values in the ranking and see how these changes effect the ranking.
3: Find distributions that show which feature values effect the ranking most.
4: Explaining phase
5: Use the distributions to sample feature values from.
6: Find the important features by observing which change in feature values effect the ranking most.
7: Return the most important features.
8: The most important features are the explanations. Return these features to the users in an understandable way.

6.2.1 Training phase

In order to find the most important features for an item, we are going to change the values of all features of that item and see how this effects the ranking. Computationally is not feasible to try as many different feature values for all features for all items in the ranking as we might want. Therefore we separate our approach in a training phase and an explanation phase. In the training phase we aim to make a distribution for each feature that describes the disruptiveness of different feature values that this feature can take on. In the explaining phase we can then use these distributions, which narrows down the search space and speeds up the computation. In this section we first describe how we measure the disruptiveness of feature values and how we decide which feature values to try (as of course even in the training phase we cannot try an infinite number of feature values). Secondly, we describe how we construct the disruptive distributions and find the most disruptive feature values that we later use when constructing the explanations for new rankings.

Step 1 - Find the disruptiveness of feature values

When measuring disruptiveness, we need to keep the assumption in mind that a feature is important, if changing the value of that feature changes the ranking a lot. Now we need to define what it means to “change the ranking a lot”. In Section 3.1 we looked into ranking similarity metrics and in particular into the Kendall’s \(\tau\) metric (Kendall, 1938) and the AP Ranking Correlation Coefficient (Yilmaz et al., 2008). In this research we choose to use the AP ranking correlation coefficient for the aforementioned reason that it focusses on the top elements
in the list. In our research this is relevant, as we deal with many articles that are scored and ranked, yet only the top 25 items are selected for the user. If we only took the top 25 into account, we would not know whether changing a feature value actually causes a particular item to be scored below position 25, yet changes at the top of the list are more relevant than changes at the bottom of the list. In most studies, rankings that obtain \( \tau \)-values higher than 0.9 are considered to be similar. As we do not have the goal to optimize rankings, we do not apply the metric this way.

**Implementation**  
Now that we have decided on our metric to measure disruptiveness we can continue to the actual implementation. Algorithm 2 summarizes the part of the training step where we find the disruptiveness of feature values. First we find the minimum and the maximum value for each feature in the training data (line 37 and 1 in Algorithm 2). We will use these values to find the most disruptive feature values for all features in the training data (line 8). We loop over all features and find ranges to sample values from (line 10 and line 27). We use the minimum and the maximum values that we found before to define the range. Moreover, we distinguish between continuous features, discrete features and features with predefined values. An example of the latter feature type would be the score the editorial staff assigns to an article. For this score they can choose between 0.3, 0.6 or 0.9. For the discrete features we select all integers between the minimum and the maximum value found, unless this number exceeds a certain bound (line 28), then we sample integers in intervals. This bound is a hyper parameter one has to choose, represented with \( \text{BinSize} \) in line 28. For the predefined feature values we only use those that are given in the data (line 30). (If these are too many, one could choose to bound this as well.) For the continuous feature values we discretise the range between the minimum and maximum values that we have found (line 33). How precisely we discretise this range is a hyper paramter that we can choose and is mostly motivated by computation time.

Now we loop through the items in the rankings of all users and change their feature values one by one, according to the feature values we have just found. We only change the feature value if the sample value is different than the current feature value, as this makes the score more sensitive (line 14). For each of these feature values we compute the \( \tau_{\text{AP}} \) (line 16), according to calculation 3.6. We keep these values and compute the average in the end (line 21). This average is what we call the disruptive score in the remainder of this section.

**Step 2 - Find distributions and their interest points**

By gradually changing all feature values that we have in our data as described above, we can find, what we call, disruptive distributions: distributions of disruptive scores per feature value. We can use these disruptive distributions to know which feature values are interesting to use and which feature values we can dismiss. That is, ideally we take feature values with low disruptive scores into account, whereas we discard feature values with high disruptive scores. (Recall that low \( \tau_{\text{AP}} \) scores represent dissimilar rankings and thus low disruptive scores describe disruptive feature values.) However, after computing the disruptive scores for each feature and its values, we do not know the disruptive distribution
Algorithm 2 Find the disruptiveness of feature values

1: function FindMinMax
2:     for each feature ∈ all features do
3:         FindMinValue(feature)
4:         FindMaxValue(feature)
5:     end for
6: end function
7: 
8: function FindDisruptiveness
9:     for each feature ∈ all features do
10:         FindSampleRange(feature)
11:     end for
12:     for each sample value ∈ sample range do
13:         for each ranking ∈ all rankings do
14:             for each item ∈ ranking do
15:                 if feature value item ≠ sample value then
16:                     change feature value
17:                     CalculateτAP(new ranking)
18:                     store τAP
19:                 end if
20:             end for
21:         end for
22:     end for
23:     average all τAP’s
24:     store average τAP
25: end function
26: 
27: function FindSampleRange(feature)
28:     if feature ∈ discrete features then
29:         return range(FeatureMinValue:FeatureMaxValue, BinSize)
30:     else if feature ∈ predefined features then
31:         return predefined values
32:     else
33:         return range(FeatureMinValue:FeatureMaxValue, BinSize)
34:     end if
35: end function
36: 
37: FindMinMax
38: FindDisruptiveness
We only know a couple of discrete values. Still we need to find the points of interest: the feature values that we want to keep for later use. We cannot do this by simply finding minima for example, as we do not know the distribution and we do not have any prior information of what this distribution could look like. Therefore, assuming a distribution and fitting our data points to this distribution can potentially lead to unwanted behaviour later on. Algorithm 3 summarizes the approach we take. Points of interest is abbreviated as pois in this algorithm.

We divide the range of disruptive scores that we have found in the previous steps in bins. The number of bins is a hyper parameter that we can choose. For each feature value for each feature we look up its disruptive score (“avg $\tau_{AP}$” in Algorithm 3) that we computed in the previous step. We compute the bin of this disruptive score (line 8). As high values for $\tau_{AP}$ mean that rankings are comparable and low values for $\tau_{AP}$ mean that rankings differ, we only keep feature values that yield the lowest average $\tau_{AP}$-values in their bin (line 9 until line 15) for the points of interest. This way we ensure that we select a different range of feature values, whereas the choice for these feature values is still motivated by their disruptiveness. Note that if feature values can only adopt a few predefined values we choose to use all these values as interest points (line 3), up until a certain number which again is a hyper parameter to tune.

Algorithm 3 Find the points of interest in the disruptive distributions

1: initialization
2: for each feature in all features do
3:   if feature $\in$ predefined features then
4:     keep predefined feature values as pois values
5:   end if
6: for each feature value $\in$ all feature values do
7:   bin = ComputeBin(avg $\tau_{AP}$)
8:   if bin $\neq$ empty AND $\tau_{AP} \leq \tau_{AP}$ in bin then
9:     put $\tau_{AP}$ and feature value in bin
10:    delete previous $\tau_{AP}$ and feature value from bin
11:   end if
12: if bin == empty then
13:   put $\tau_{AP}$ and feature value in bin
14: end if
15: end for
16: end for
18: return kept feature values as pois values

6.2.2 Explaining phase

Now that we have found the disruptive distributions and the points of interest, we can proceed to the explaining phase, where we find the most important features for all items in the ranking. Algorithm 4 summarizes the explaining phase of LISTEN.

For each new ranking that comes in, we change all of its feature values (i.e., all feature values for all items in the ranking) according to the points of interest.
Then we compute the AP ranking correlation coefficients for the new ranking that arises from changing this feature value. We compute the average AP ranking correlation coefficients for this feature (line 5 until line 7), in the same fashion as we described for the training step of our approach (Algorithm 2). However, at this stage we distinguish between points of interest that decrease the ranking score (line 8) and points of interest that increase the ranking score (line 11), in order to be able to clearly communicate the effect of certain features to the user. In the next step we calculate the average AP ranking correlation coefficients (lines 15 and 16). The features that were most disruptive, according to this method, are selected as most important features and used as explanations (line 18). How many features one reports is again a hyper parameter to tune. As a final step we normalize the labels (line 19), so that we keep the relative importances of each important feature in comparison to the other important features. Another option would be to define binary labels, which simply state whether a feature was important or not. We choose the continuous approach as that allows for a more detailed explanation. Only the upwards pushing labels are used in the communication to the users.

Algorithm 4 Make labels with importance scores

1: for each ranking ∈ all rankings do
2:     for each item ∈ ranking do
3:         for each feature ∈ item do
4:             for each pois value ∈ all pois values for feature do
5:                 if pois value ≠ feature value then
6:                     change feature value
7:                 end if
8:                 CALCULATE$\tau_{AP}$(new ranking)
9:                 if new item ranking score < old item ranking score then
10:                    add $\tau_{AP}$ to upwards pushing $\tau_{AP}$
11:                 end if
12:             end if
13:         end for
14:     end for
15:     average upwards pushing $\tau_{AP}$ and add to upwards pushing label
16:     average downwards pushing $\tau_{AP}$ and add to downwards pushing label
17:     end for
18: choose most important feature values
19: normalize labels
20: end for

6.3 Q-LISTEN: Speed ups with neural networks

Both explanations methods that we use, mLIME and LISTEN, are very time consuming as they need to sample many feature values and do computations for all these samples. Because of this property both methods do not scale very well. Ribeiro et al. (2016) mention this as one of the shortcomings of their system.
Scalability is an essential property if one wants to implement an explanation system that runs in production in real time.

Recall the Blendle pipeline, that was described in Chapter 4. Theoretically we could make the pointwise explanations at the time when the article enters the pipeline. This allows us to take a bit more time to construct the explanations, as articles can come in during the entire day. However, the majority of news articles comes in the morning. Moreover, we specifically wrote “theoretically”, because in practice, the pipeline does not allow for this. However, if we really wanted, we could, by adjusting the pipeline here and there. The listwise explanations on the other hand, can only be made when the final ranking for a user is made. This gives us much less time than when we could compute the explanation when the article comes in, as at the time of ranking, we want to send out the users’ personalized rankings as fast and as soon as possible. As, in the end, we deal with news articles.

We could deal with this problem by decreasing the number of sample values that we use for our explanations, or by implementing some heuristics. However, this decreases the faithfulness of the explanations and it does not speed up the system as much as we want. Therefore we need to come up with a different solution.

In order to solve the problem we train a neural network, that approximates the function that expresses the importance of features in a ranking. This is a function we do not know and therefore neural networks are especially suited for this task. Moreover, once we have trained a network, at most we need to do a number of matrix multiplications and add some non-linearities. This is scalable to run in production in real time. We refer to this approach as Q-LISTEN and Q-mLIME.
The input and the output of our networks are different for Q-mLIME and Q-LISTEN. This is shown in Figure 6.1. For Q-mLIME, a single item vector, with its corresponding feature values, enters the network and an output vector with importance scores per feature value is returned (figure 6.1a). On the other hand, for Q-LISTEN an entire ranking enters the network and the network needs to learn the dependencies between feature values in the entire ranking (figure 6.1b). As our input space is relatively small we choose to use an approach that was also used in image recognition in the early days and has proven to work well. For the listwise method we flatten the input data, such that our input matrix becomes an input vector where all items and their feature values are placed below each other. We have considered using Convolutional Neural Networks (CNNs) for the listwise approach, as, intuitively, the input data is comparable to the input data in image classification, where CNNs work extremely well. However, one essential difference between images and our input data, is that pixels in parts of the image have a clear relation, whereas we cannot assume such a relation in our input. Therefore we choose to flatten our input data and train a Multilayer Perceptron (an MLP). All hyper parameters of the MLPs that we use are described in Chapter 7.

Note that the input space could potentially blow up quickly if many more features with many potential feature values are added. The networks that can be used then have to be a lot more advanced and this is something we leave for future work.

6.4 Dealing with diversification

There are two steps in the pipeline where diversification plays an important role: the ranking diversification, that was described in Section 4.1 and reason diversification, that was mentioned in Section 4.2. In this section we describe how we deal with both diversification types in the current research.

**Ranking diversification**  Recall that a user’s ranking is diversified to ensure that it contains a wide variety of different articles types. This is done after the articles have been ranked solely based on ranking scores. The ranking diversification causes items that would not have been ranked in the user’s top $N$ now do find their place in the user’s personal selection. Self-evidently this also causes other items, that would have been selected for the user’s top $N$ without diversification taking place, to fall outside the user’s personal selection. The method that we have described so far does not deal with these new items in the ranking, as the pipeline is only trained to use the item-user-features to find explanations. It does not take the features that are used for the diversification into account. As we investigate the explainability of ranking algorithms in this research and this ranking diversification is not strictly ranking related, we choose to not directly address this diversification in the current research. For research purposes this would suffice, yet as we run our system in production on real Blendle users, it is impossible to completely ignore the fact that diversification is part of the pipeline. Therefore we choose to add special diversification reasons. Before the diversification step we compute all reasons in the way we have described in this chapter. After the diversification step we look for items in in the final
A ranking that have scores that do not fall within the user’s top \( N \). These items are then explained with a diversification reason. This way we ensure that we stay faithful to the model.

The easy alternative, training the network on more items in the ranking, would not ensure faithfulness as after all, if items are placed in the ranking because of the diversification algorithm then that is the only reason why this item was shown to the user. A perhaps more faithful alternative would be to include the features that are used for the diversification in the explaining pipeline. We discuss the theoretical implications of this method in more detail at the end of this thesis, in Chapter 9.

**Reason diversification**  In searching for the balance between faithfulness and user friendliness we are always focussing on faithfulness in this research. However, we make one concession that makes the algorithm slightly less faithful, yet a lot more user friendly. This concession is the reason diversification. Our algorithm computes most important features and returns \( N \) of these. By default we choose \( N = 3 \) in this research. In principle, we show the most important reason to the user. However, we also want to show a diverse spectrum of reasons. Therefore we choose to diversify our reasons. That is, we compute the similarity of a computed reason and the reasons that are already reported. We choose the reasons with the lowest similarity. As we only take the three most important features into account as potential reasons, these reasons are still faithful and yet users do never see the same reason over and over again. Reasons are assumed to be similar when they are from the same type or when they include the same provider or channel.

### 6.5 Communication to the user

As for now we have constructed reasons in a way that we cannot communicate to users — we cannot assume a vector or a matrix with importance scores to be understandable. The goal of this research is not to automatically generate a human understandable format from these important scores by using approaches such as caption generation (Xu et al., 2015, e.g.). Instead, we manually construct a one-to-one mapping between features and human understandable explanations. From our user study (that we described in Chapter 5) we could conclude that users did not have a strong preference for either one of the designs we showed them. For the current application we decide to write copy for all feature values. In Table 6.1 all features and the corresponding copy are summarized. Note that the copy of these features are the English translations of the Dutch sentences that we use in this research. All explanation texts are carefully constructed together with Blendle’s copy writer to match Blendle’s house style. We decide to write no copy for the ‘item number of images’ feature, as that feature had a negative weight at the time that this research was conducted, i.e. the more images the lower the score for that item. For users it is somewhat confusing to read a reason that captures exactly this. We link the features to their copies and we only return the copy to the users. As far as the design is concerned, we show the explanation for an article after hovering over an “i”-icon on the introduction card of the article on blendle.com. We decide to show only one reason to the
Table 6.1: Features that are used in the Blendle pipeline and the copy that is shown to the user.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>item rating score</td>
<td>“Our editors think this story is worth your time.”</td>
</tr>
<tr>
<td>item pick probability</td>
<td>“Our in-house robots picked this out as a good story.”</td>
</tr>
<tr>
<td>item number of images</td>
<td>No copy</td>
</tr>
<tr>
<td>item channel followed by user</td>
<td>“Because you’re interested in channel X.”</td>
</tr>
<tr>
<td>item provider followed by user</td>
<td>“Because you follow provider X.”</td>
</tr>
<tr>
<td>newsletter purchased channel score</td>
<td>“Because you read about X more often.”</td>
</tr>
<tr>
<td>newsletter purchased provider score</td>
<td>“Because you read a lot from X.”</td>
</tr>
<tr>
<td>user item negative channel feedback</td>
<td>“Because we think channel X can be interesting for you. Not true? Try the broken heart.”</td>
</tr>
<tr>
<td>user item negative provider feedback</td>
<td>“Because we think provider X can be interesting for you. Not true? Try the broken heart.”</td>
</tr>
<tr>
<td>must read</td>
<td>“The editorial found this one of the best pieces today. No matter your preferences.”</td>
</tr>
<tr>
<td>diversification 1</td>
<td>“Because you want something different every now and then.”</td>
</tr>
<tr>
<td>diversification 2</td>
<td>“Because we want to show you all kinds of different articles.”</td>
</tr>
</tbody>
</table>

user. This mainly has to do with how much information fits on the article card. Figure 6.2 gives an impression of what the design looks like.
**Figure 6.2:** Impression of how a reason is shown to the user. The black box on the second card states: 'Why this article? - Because you often read from Trouw.'
Chapter 7

Experimental setup

In this chapter we describe the experimental setup that we will use to answer research question 4 and 5. We start with a description of the data. Then we describe the experimental setup for the mLIME baseline, then for LISTEN and finally for the neural networks that we use for efficiency reasons.

7.1 Data

We extract around 30Gb of feature data of Blendle users. Amongst others, this data contains all feature values for all items for approximately 5500 users. This includes both active and less active users. It is important to have a good mix between these two groups, perhaps even slightly biased towards active users, as non-active users will all have very similar feature values. From this data we select a number of users that we use in the training step of our explaining systems. For the mLIME baseline we select 275 users (active and less active) for our training data and for LISTEN we select 100 users (active and less active), as the training step of LISTEN is computationally slightly heavier. We use the rest of the data for the explaining step. After the explaining step this data is again divided into training, validation and test data to build the speed up step with the neural networks.

7.2 mLIME baseline

In this section we describe the implementation details of the mLIME baseline. We especially focus on how we have implemented our binning function that we have introduced in Section 3.2.2 and that serves to treat our ranking function as a classification function. We also describe how we use mLIME to construct data in such a format that it can be used to train a neural network on in more detail.

From ranking function to classifier  Recall from Chapter 3 that LIME first trains an explainer and that it then uses this explainer to return the most important properties of a data point that lead to the classification of that data point in its class. We use the training data that was described in Section 7.1 to
train the explainer. We use the standard LIME implementation to do so, except for the fact that we need to make sure that our ranking function is treated as a classifier. Therefore we rewrite the function that LIME uses to predict the class probabilities. In our implementation we bin all scores, as described in Chapter 3. We use the training data to decide on the number of bins. Figure 7.1 shows how the data points in the training data are scored. We see six clear regions. Because of this distribution of the data, we choose to center our bins around these six regions. Per region, we make two bins. That means that we divide our data over twelve bins in total. More bins per region make mLIME less reliable and cause the algorithm to return zero values for all importance scores. Using less bins per region is not preferred, as we want to capture as much variance in our data as we can. Table 7.1 gives an overview of the mean and standard deviation of scores in each region in the training data and how often this region occurs.

Using the mLIME explainer Once we have this explainer, we use this explainer to make training, validation and test data for a neural network. We feed our data points (i.e. feature vectors) to the mLIME explainer and let the explainer come up with the explanations. An explanation could look like (0.5, feature value X > value Y) or (0.23, value Y ≤ feature value X ≤ value Z). For

---

**Figure 7.1:** Distribution of data points and their scores.

**Table 7.1:** Mean, standard deviation and occurrence regions.

<table>
<thead>
<tr>
<th>Mean score</th>
<th>Standard deviation</th>
<th>Occurrence (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>939</td>
<td>0.86</td>
<td>0.69</td>
</tr>
<tr>
<td>−8</td>
<td>0.72</td>
<td>4.48</td>
</tr>
<tr>
<td>−3595</td>
<td>16.46</td>
<td>2.87</td>
</tr>
<tr>
<td>−4539</td>
<td>14.42</td>
<td>27.9</td>
</tr>
<tr>
<td>−8144</td>
<td>0.76</td>
<td>6.72</td>
</tr>
<tr>
<td>−9091</td>
<td>0.70</td>
<td>57.24</td>
</tr>
</tbody>
</table>
the first example, this would mean that the fact that the value of feature $X$ is bigger than a certain value $Y$ has a weight of 0.5 for the explanation. An mLIME explanation for a single data point is a combination of multiple of these explanations: different features and feature values are combined into a single explanation. In order to be able to feed this explanation to a neural network we need to rewrite it. As stated in Chapter 6 we construct continuous labels and we distinguish between upwards pushing features and downwards pushing features. mLIME explanations with positive weights are added to the upwards pushing label, whereas mLIME explanations with negative weights are added to the downwards pushing label. For all labels we choose the three most important features and normalize their importance scores.

Now we have all data that we need to train our neural network on. We discard all data points that fall outside the user’s top $N$, whereby $N = 25$ in the current research, equal to the number of items in a user’s personal selection. We decide to do so, because those items are not shown to the user. Only using the top $N$ increases the final accuracy of the network with roughly 10%. This can be explained by the fact that the feature values and corresponding scores are noisier in the lower scoring regions. For a detailed description for the setup of the neural networks we refer to Section 7.4.

7.3 LISTEN

In this section we describe the implementation details of LISTEN. In particular, we describe the construction of the disruptive distributions in more detail and we show what these distributions look like. We also look into the implementation details of how we make the training data for our neural networks for LISTEN.

Making disruptive distributions and finding their interest points Given that we have the data in the format that was described in Section 7.1, the rest of the LISTEN pipeline proceeds as follows. In order to finish the training phase, we need to find disruptive distributions. We described the method to do this in the previous chapter. In this section we give the hyper parameters and we show the plots of the disruptive distributions.

For all features we find the minimum and maximum values in the training data. For the continuous features we divide these ranges in 20 sample points. For the discrete features we divide these ranges in discrete sample points, with a maximum of 20 as well. In the current application we do not find discrete ranges that exceed the bound of 20, yet for other applications one could consider sampling every other integer in the range if the bound of 20 is exceeded. For the predefined sample values we use all values that we find in the data, again unless a bound of 20 is exceeded. This is never the case in the current research. We choose the value of 20 as by inspection we find that this is a good threshold between computation speed and accuracy.

By using Blendle’s log-linear ranking function, we make rankings for all data, i.e. for the users whose data points are used in the training phase as well as for the users whose data points are used in the explaining phase. We only use the rankings in the training data set to make the disruptive distributions. Now we sample from the ranges we just created in the fashion that was described
Section 6.2.

Per disruptive distribution we make 20 bins to find the points of interest. Recall from Section 6.2.1 that we bin the $\tau_{AP}$-values and not the feature values. In the explaining phase we use the points of interests to replace the feature values with and to find out which features are most important. Figure 7.2 shows the disruptive distributions with their points of interests for all features. For the predefined sample values we only plot the points of interest, for the other feature values we also plot the disruptive distribution underneath.¹ We do not find the same number of interest points for all features. This is because for some features a small difference in feature value can already change the ranking. This is for example the case with the feature values 5 and 6. These feature values are comparable in nature: they both represent a purchased score (either for the provider or the channel) and many different feature values occur for this feature in the data. On the other hand, changing the feature values for the pick probability (feature 2) does not change the ranking as quickly, but there are points that are clearly more disruptive than other points. There are also less values that occur in the data for this feature. The disruptive distributions for feature values 7 and 8 also behave very similarly. These are the disruptive distributions of the negative feedback features. These plots are partly explained by the structure of the data for this feature and partly by a mistake in our code, that in the end does not effect our results.² The plot for feature 4 could be explained by the fact that there are many items in the data that have a 0.9 score for this feature and often these items are ranked in the higher regions of the ranking. Then adding 0.9 for this feature for items that did not have a score of 0.9 does not change the ranking drastically, as there are still many items that are scored higher. However, adding 0.6 or 0.3 for items that had a 0.9 score, may cause that the item was now scored lower than the item below it, decreasing the $\tau_{AP}$ score.

¹Note that by accident also some non-discrete values were used for feature value 1, however, as these values were all very close to discrete values and we are using a log-linear ranking function this does not effect our results.

²The features for the negative feedback were just added to the Blendle scoring pipeline before training our listwise model. This causes that many users have not given negative feedback yet and that the users that have given negative feedback have scores extremely close to 1.0 (the score that is given when users have not given any negative feedback). Now, as many users only have scores of 1.0 for the negative feedback, adding 1.0 could not influence the ranking in any way. This edge case (it only occurs for these two features, with the current data set) caused the algorithm to add 0.0 for the $\tau_{AP}$-values, which makes the score go down. In the end, this does not influence the rest of our method, as scores of 0.0 and 1.0 are found as points of interest, which are in fact the points that we would like to find. For the final importance scores that are computed in the explaining phase (explained in more detail in the next paragraph), we use a different part of our code, that does not include this mistake, so the final results are not influenced by it. We also see that the $\tau_{AP}$ values before 1.0 are all similar. That is because of the structure of the data. As stated, all negative feedback values were very close to 1.0 or were 1.0 and therefore, adding lower features values changes the ranking similarly for all documents.
Feature 0
item provider followed by user

Points of interest

Feature 1
item number of images

Feature 2
item pick probability

Feature 3
item channel followed by user

Feature 4
item rating score

Feature 5
purchased channel score
The explaining phase We use the rest of our data for the explaining step. The pipeline of the explaining step was described in Section 6.2.2. Again we choose to make continuous labels for our neural network and we normalize the importance values. We choose to learn three important features per item and we distinguish between upwards pushing labels and downwards pushing labels.

7.4 Q-mLIME and Q-LISTEN

In this section we give all hyper parameters that we use during the training of our neural networks. We use TensorFlow for this task (Abadi et al., 2015). For both Q-mLIME and Q-LISTEN we divide our data in training (70%), validation (15%) and test (15%) data. The hyper parameters for both models are given in Table 7.2. Moreover, we use batch normalization, as described in Section 2 and a final linear output layer.

A few parameters stand out: in the end, we do not apply weight regularization, as the weight regularization strength is 0. We tested several settings, yet smaller
Table 7.2: Parameter values multi-layer perceptrons for Q-mLIME and Q-LISTEN.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Q-mLIME</th>
<th>Q-LISTEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning rate</td>
<td>$2e^{-3}$</td>
<td>$2e^{-4}$</td>
</tr>
<tr>
<td>Weight regularization</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Weight regularization strength</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Batch size</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Training steps</td>
<td>5000</td>
<td>6000</td>
</tr>
<tr>
<td>Dropout rate</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>Weight initialization</td>
<td>Xavier</td>
<td>Xavier</td>
</tr>
<tr>
<td>Activation function</td>
<td>ReLU</td>
<td>ReLU</td>
</tr>
<tr>
<td>Optimizer</td>
<td>ADAM</td>
<td>ADAM</td>
</tr>
<tr>
<td>Number hidden layers</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>Hidden nodes per layer</td>
<td>100 - 100 - 100 - 100</td>
<td>100 - 100 - 100 - 100 - 100</td>
</tr>
<tr>
<td>Loss function</td>
<td>Mean Squared Error</td>
<td>Mean Squared Error</td>
</tr>
</tbody>
</table>

regularization strengths did not decrease the accuracy on the validation set. On the other hand, larger regularization strengths decreased the output values to values very close to zero. These values are less stable as they are more sensitive to small perturbations and numeric underflow. Therefore we choose to discard the weight regularization, also given that we apply other methods to avoid overfitting. Moreover, we need an additional layer to learn the mLIME function in comparison to the LISTEN function. This underlines our hypothesis that mLIME is less reliable and that the underlying function is more difficult to learn because of that. However, the learning rate for Q-mLIME is higher than the learning rate for Q-LISTEN and the number of training steps is lower for Q-mLIME than for Q-LISTEN. This shows that the Q-mLIME method converges faster than Q-LISTEN. (Even though in Chapter 8 we will see that Q-mLIME converges to a lower accuracy than Q-LISTEN.)

7.5 Evaluation

As mentioned in Chapter 2 the evaluation of an explanation system is not straightforward. Apart from the evaluation of our neural networks, we distinguish two other types of evaluation methods: an offline evaluation of the explanation systems and an online A/B-test to investigate how users interact with different types of explanation systems. Below we describe each of these methods in more detail.

**Offline evaluation of the explanation systems** First we need to design an offline evaluation method to evaluate whether LISTEN does what we expect it to do. In order to test this, we construct dummy data in the lines of the example that we gave in Chapter 1. On this dummy data, with a dummy scoring function, we know what to expect, namely precisely this behaviour as we described in Chapter 1. Moreover, we compare the outputs of mLIME and LISTEN on real data.
Table 7.3: P-values for the differences between groups before the experiment starts.
We compare all groups with each other and use $\alpha = 0.05/3 = 0.0167$ as significance level for these comparisons. We also compute the p-value for the groups that scores highest on average. All values are computed with a randomization test.

<table>
<thead>
<tr>
<th></th>
<th>Heuristic vs Q-LISTEN</th>
<th>Q-LISTEN vs Q-mLIME</th>
<th>Heuristic vs Q-mLIME</th>
<th>Q-LISTEN scores highest</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>0.355</td>
<td>0.457</td>
<td>0.883</td>
<td>0.468</td>
</tr>
</tbody>
</table>

A/B testing As a last evaluation metric we test how users interact with a system that provides different types of recommendation reasons. We compare the Blendle heuristic reasons, the reasons provided by Q-mLIME and the reasons that are computed with Q-LISTEN. In order to make this comparison we run an A/B-test, or rather, an A/B/C-test in this case, on real Blendle users in the real Blendle environment.

We run the A/B-test for two weeks on all Blendle users that receive a personalized news selection. For competitiveness reasons we cannot reveal the precise number of users. The reasons are implemented on blendle.com and are shown when a user hovers over an “i”-icon. The first five times a user enters the website, or until a user has seen a reason, an info-box is shown to alert users about this feature. Figure 7.3 and 7.4 show examples of what users see during the test.

We divide the users in three groups. One group is shown the default heuristic reasons that were shown by Blendle before. The second group is given the Q-mLIME reasons and the third group is given reasons produced by Q-LISTEN. To make sure we include all users, we assign 34% of the users to the heuristic group and we equally divide the other 66% over the Q-mLIME group and the Q-LISTEN group. We normalize for this in our evaluation. We implement stratified sampling (see Section 3.3) to divide users over groups. Figure 7.5 shows the reading behaviour of the users in the different groups eighteen days before the start of the experiment. Again, for competitiveness reasons we cannot reveal the raw numbers, so we report all reads as percentage of the total number of reads.

We see that the counts in the heuristic groups are indeed slightly higher than the counts in the other two groups. This is what we expect from our 34 − 33 − 33 division. If we correct for this, we see that the Q-LISTEN group reads most before the start of the experiment, yet we do not see a significant difference between the three groups, see Table 7.3. Taking the Bonferonni correction into account, we use $\alpha = 0.05/3 = 0.0167$ as significance level for the comparisons between two groups. For the comparison of the highest scoring group we do not need to correct for multiple comparisons, so we use $\alpha = 0.05$ as significance level. We compute the p-values with a randomization test, that was described in Section 3.3. We use the two-tailed version for the computation of the statistics of the differences between the groups and the one-tailed version for the computation of the statistics for the highest scoring group.
Figure 7.3: Example of the infobox that the user sees to make the user aware of the possibility to ask for a reason. The black box states: 'Here you can see why we recommend this piece.'

Figure 7.4: Example of a reasons that a user can see during the test after hovering over the 'i'-icon. The black box on the third card states: 'Why this article? - Because we think that de Volkskrant can be interesting for you. Not true? Try the broken heart.'
Figure 7.5: Number of articles opened per day in the different groups. Normalized over all days and all groups.
Chapter 8

Results

In this section we answer our last two research questions, namely, (1) how do we build an explanation system that produces faithful explanations for the outcome of a ranking algorithm, yet is scalable and (2) does the reading behaviour of users who are provided with model-agnostic listwise explanations for a personalized ranked selection of news articles differ from the reading behaviour of users who are provided with heuristic or pointwise explanations for a personalized ranked selection of news articles? We also give examples of what the explanations look like for the users.

8.1 RQ 4 - Are our explanations faithful and is the method scalable?

In this section we will look into the faithfulness and scalability of LISTEN.

8.1.1 Are our explanations faithful?

As a first step we want to find out whether LISTEN produces faithful explanations. We test this with the ranking function and the ranking that we introduced in Chapter 1 and that we repeat here for clarity reasons. The simple linear ranking function is given by

\[
score(x_0, x_1, x_2) = 0.2x_0 + 0.3x_1 + 0.5x_2,
\]

and the ranking is given by

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d_0)</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(d_1)</td>
<td>0.5</td>
<td>0.5</td>
<td>0.75</td>
</tr>
<tr>
<td>(d_2)</td>
<td>1</td>
<td>0</td>
<td>0.75</td>
</tr>
</tbody>
</table>

Recall from Chapter 1 that we want LISTEN to find that for both item \(d_0\) and \(d_2\) feature \(x_1\) was most important. We have two steps that we need to validate.
First, we need to verify whether the general idea of changing the feature values result in the correct behaviour. Secondly, we need to verify whether using the interest points that we find by making the disruptive distributions, also lead to the expected behaviour.

**Changing the feature values**  First we test our approach of changing the feature values, without selecting points of interest. We change the feature values in their entire domain, with steps of 0.01. We find exactly the behaviour we were expecting. The $\tau_{AP}$ values for the ranking are found to be

<table>
<thead>
<tr>
<th>$d_0$</th>
<th>$x_0$</th>
<th>$x_1$</th>
<th>$x_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>0.83</td>
<td>1.0</td>
</tr>
<tr>
<td>0.64</td>
<td>0.67</td>
<td>0.99</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>0.83</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>

Now, looking at the first and the last items in the ranking we see exactly what we predicted in Chapter 1. The ranking cannot change by changing feature $x_0$ or $x_2$ in their valid ranges. The ranking can change if we change value $x_1$ and this causes $x_1$ to have the lowest $\tau_{AP}$ value.

**Using disruptive distributions**  We also need to validate whether our approach of using the interest points that we have found by making the disruptive distributions is faithful. Therefore we construct dummy ranking data. We use the three features that were introduced above and their domains. We also use the same scoring function. We make data points (i.e. ‘items’) by randomly sampling feature values for each feature. We sample from a range with steps of 0.01.

We want to find out whether the number of users in our data and the number of data points per user influence the results. Therefore we make data for multiple numbers of users and a range of data points per user. For the number of users we choose from $[5, 10, 20, 100]$ and for the number of data points we choose values from $[5, 10, 20, 40, 60, 80, 100, 120, 150]$. We make all possible combinations with it. We make disruptive distributions, find interest points and compute the most important feature values for our known sample ranking based on these interest points. We compute the accuracy, i.e., how often our approach returned the correct feature values. Because our approach is not deterministic, as we randomly choose values for the feature values in the data points, we compute the accuracy twenty times per setting (i.e., we are constructing twenty data sets per setting) and average these.

Figure 8.1 shows the accuracy scores per number of users per number of data points. Figure 8.2 shows the average accuracy scores with their standard deviations. Even though it does not make sense to have an accuracy above 1.0 we still plot these values, to be able to show the intervals. We see no vital differences between different settings. We do see that we do not have a 100% score at all times. These lower scores are most often caused by the reason returned for $d_1$. The disruptiveness of feature $x_0$ and $x_1$ are quite similar for $d_1$ and sometimes $x_0$ is chosen as the most important feature value.

In our real data, i.e. the Blendle data, the disruptiveness of the different features can also be very similar. This Blendle data however, is a lot more structured.
than the random data that we have constructed for this dummy experiment, as per feature, some feature values just occur a lot more than other feature values. That is, we can rely on our disruptive distributions a lot more, as long as we make sure that we make these disruptive distributions based on a representative sample of the data. In our case, this means that we need to include active users and less active users. Secondly, if features have very similar disruptive values, these features are of very similar importance in the end. This also reduces the impact on faithfulness reporting the feature value that was actually slightly less important than another feature value, especially as we calculate more than one important feature as reason.

Comparing the mLIME and LISTEN

In this section we compare the reasons that are produced by mLIME and reasons that are produced by LISTEN on real data. In general we can conclude that the reasons that are returned by both methods partly overlap and partly differ. This is an intuitive result, as we look at the $N$ most important features. E.g., it can be expected that some of the features that play a very important role in the increase of the score of an individual item, are also responsible for pushing an item up the ranking. However, we have seen that the behaviour of features can be different on a pointwise level and a listwise level. Therefore it is also expected that we find many differences between the two approaches. In Table 8.1 we present some examples from the data. The examples are taken from the selection of different users. For each item, we mark the three most important features and we highlight the ‘winner’. Note that mLIME often returns less than three important features. This is because the mLIME approach tends to give a lot of (negative) weight to features that actually push the score down and here we present the features that push the score up, as we have also used this in our communication to the user. Sometimes features have the same importance.
Figure 8.2: Average accuracy and standard deviations when using disruptive distributions for different number of users and different number of data points.
Table 8.1: Most important and winning features computed by both methods. The winning feature is shown blue, the other important features are shown red and the rest of the features are shown grey.

<table>
<thead>
<tr>
<th>Position item in ranking</th>
<th>mLIME or LISTEN</th>
<th>$f_0$</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mLIME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LISTEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>mLIME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LISTEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>mLIME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LISTEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8.2: Final loss and accuracy scores for different methods for train, validation and test set.

<table>
<thead>
<tr>
<th></th>
<th>Loss</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Q-mLIME</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training</td>
<td>0.00316</td>
<td>0.647</td>
</tr>
<tr>
<td>Validation</td>
<td>0.00569</td>
<td>0.667</td>
</tr>
<tr>
<td>Test</td>
<td>0.00531</td>
<td>0.659</td>
</tr>
<tr>
<td><strong>Q-LISTEN</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training</td>
<td>0.00135</td>
<td>0.986</td>
</tr>
<tr>
<td>Validation</td>
<td>0.00121</td>
<td>0.985</td>
</tr>
<tr>
<td>Test</td>
<td>0.00139</td>
<td>0.987</td>
</tr>
</tbody>
</table>

scores, then we randomly choose which feature is returned as the most important one. Table 8.1 also nicely shows that the reasons returned by mLIME and by LISTEN partly overlap, but not entirely.

**Faithfulness of neural network speed-ups** Figure 8.3 shows the development of the loss and accuracy of the neural networks on the train and validation data for both Q-mLIME and the Q-LISTEN. Table 8.2 summarizes the final loss and accuracy values on the train, validation and test set, also for both methods.

We see that the final accuracy of Q-mLIME is a lot lower than the final accuracy of Q-LISTEN. This underlines our hypothesis that the mLIME is less trustworthy for the current problem than LISTEN, as apparently the underlying function of mLIME is more difficult to learn than the underlying function of LISTEN. For Q-LISTEN we do not achieve a 100% accuracy either. Yet the speed-up that is achieved by using this method makes it worth using it (otherwise no listwise reasons could be generated at all).
Figure 8.3: Development of loss and accuracy on train and validation set for Q-mLIME and Q-LISTEN.
8.1.2 Is our method scalable?

Now that we have seen that the networks are very well able to learn the latent explaining function, at least for the listwise approach, it is worth investigating to what extent they improve the speed of the pipeline. On a simple notebook (8GB RAM, i5-5200U processor) it takes around 30 seconds to generate a listwise reasons for an item, given that the disruptive distributions were already made. Using the neural network, this only takes around 1 millisecond. Given that we have to make millions of reasons at production time, this is a speed-up that changes the run time from ‘infeasible to run in production’ to ‘perfectly fine to run in production’.

8.2 Some examples

Before we continue to our last research question and present the results of our A/B-test, we show some examples of what users are presented with and we elaborate on that. These examples are taken from the personalized selections of Blendle employees in the Q-LISTEN group.

The first example is given in Figure 8.4. The user is presented with an article about science. The reason that is given says ‘Because we think Science can be interesting for you.’ This user indicated to like science and to be slightly surprised by this reason. Why would that have to be a guess? Until this user looked into the preference settings and noticed not to have added Science as a personal preference during the onboarding.

Now we give an example that shows that focusing on faithfulness does not always generate the most intuitive reason. Figure 8.5 shows a recommendation that states that this article from de Volkskrant is recommended to the user because the algorithm expects that ‘opinion’ is an interesting channel for this user. However, this user follows this channel: of course this user finds the channel interesting. This reason is generated from the feature ‘user doesn’t dislike channel’, that captures how much negative feedback a user has given on this channel. Naturally, if a user follows a certain channel, apparently he or she enjoys the channel and the chance that he or she gives negative feedback on this channel is smaller than if he or she would not follow this channel. Therefore, the reason can still be faithful, even though it may not be very intuitive for a user.

Figure 8.6 and 8.7 show two other articles from the newspaper de Volkskrant. The reason that is given in Figure 8.6 states that this article was recommended to the user because this user reads more often about the United States, which is the channel of this article. This particular user indeed reads about the United States more often. The reason for the article that is shown in Figure 8.7 states that this user reads more often from de Volkskrant. Indeed, this user reads a lot from this newspaper. These last three examples (figures 8.5, 8.6 and 8.7) all come from the same user and even though they were all from de Volkskrant, they all show different recommendation reasons. This gives an impression of the variety of reasons that is shown to users.
Figure 8.4: Recommendation of a Science piece. Recommendation reason translation: ‘Because we think Science can be interesting for you. Not true? Try the broken heart.’
Figure 8.5: Recommendation of an article about opinion. Recommendation reason translation: ‘Because we think Opinion can be interesting for you. Not true? Try the broken heart.’
Figure 8.6: Recommendation of an article about the United States. Recommendation reason translation: ‘Because you read more often about the United States.’
Figure 8.7: Recommendation of an article from ‘the Volkskrant’. Recommendation reason translation: ‘Because you read many articles from the Volkskrant.’
8.3 RQ 5 - How do users interact with different reason types?

After two weeks we analyse the results of the A/B-test, that was described in Section 7.5. Figure 8.8 shows the reads of all Blendle in the heuristic group, the Q-mLIME group and the Q-LISTEN group over these two weeks. Even though only the users that receive a personalised bundle and read from blendle.com are included in the experiment, all other users are automatically divided into groups as well. They just do not see any difference. To get a feeling for the reading behaviour of all users over the period the experiment was live, we start with a brief analysis where we include all users. Again we do not report the raw data for competitiveness reasons.

We normalize for the fact that the heuristic group has slightly more users (namely 34% of the users instead of 33%). Then the users in the heuristic group have read most over the fourteen days the test was live. A randomization test as described in Section 3.3 reveals that this is a non significant result ($p = 0.955$). When we compare the total reads in all groups we find no significant differences between any of the groups either. Table 8.3 summarizes this.

Now we extract the users that receive a personalized selection from the total number of users and that have opened the articles at blendle.com, as only those users can potentially be effected by the personalized reasons. Figure 8.9 shows...
the reads of these users in the heuristic group, the Q-mLIME group and the Q-LISTEN group over these two weeks. Table 8.3 shows that again the differences between groups were not significant.

**User interaction with reasons** We also investigate how many users have seen reasons and how that effected their behaviour. Naturally, again we only compare the users with a personalized news selection that have read the articles at blendle.com. Again all results are normalized for the fact that the heuristic group includes 34% of the users and the Q-mLIME and the Q-LISTEN group both include 33% of the users. We investigate the number of users that open articles within two minutes after they have seen a reason. Moreover, we count the number of users that hover the reason icon after they have read the article, that is, within twenty minutes after opening the article. The results are summarized in Table 8.4 and 8.5. Again we do not find any significant differences between the three groups. Users ask for reasons almost as often shortly before reading the article as after reading the article. The rest of the reasons made users decide either not to read the article or read the article beyond the time frames that we measure here.

To find out whether users might interact differently with reasons at the beginning
Table 8.5: P-values for comparison between the number of reasons seen in all groups.

<table>
<thead>
<tr>
<th></th>
<th>Heuristic vs Q-LISTEN</th>
<th>Q-LISTEN vs Q-mLIME</th>
<th>Heuristic vs Q-mLIME</th>
<th>Highest group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of reasons seen</td>
<td>0.507</td>
<td>0.736</td>
<td>0.707</td>
<td>heuristic 0.640</td>
</tr>
<tr>
<td>Reasons seen within two minutes before opening the article</td>
<td>0.298</td>
<td>0.065</td>
<td>0.575</td>
<td>Q-mLIME 0.139</td>
</tr>
<tr>
<td>Reasons seen within twenty minutes after opening an article</td>
<td>0.880</td>
<td>0.663</td>
<td>0.607</td>
<td>heuristic 0.819</td>
</tr>
</tbody>
</table>

of the experiment than at the end of the experiment, we plot this development in Figure 8.10. It could be that users interact more with reasons at the beginning of the experiment, because at that time they were made aware of the feature. However, even though we do see differences over time, in general the number of reasons that is shown per group is quite stable, so users do not seem to be influenced by the “newness” of the reason feature.

We also compare which reasons were seen most often by users. This is summarized in Figure 8.11. From these figures it stands out that the must read reason (reason 10 in the heuristic group and reason 11 in the Q-mLIME and Q-LISTEN group) is clicked most often. The must reads are shown at the beginning of a user’s selection, which can bias these results. Furthermore we see small differences between the reasons that were seen in the Q-mLIME group and in the Q-LISTEN group.

**Figure 8.10:** Reasons seen per group per day. Normalized by total number of reasons seen in entire period per group. Only users with personalized selection included.
given that we have not seen any significant differences in user behaviour, this is most likely due to the different reasons these two approaches produce.

We do not see any differences in reading behaviour or interaction with the reasons between the different groups. However, there may be differences between the users that do interact with the reasons and users that do not interact with the reasons. In fact, if we find that users that do interact with the reasons tend to read significantly more on average than users that do not interact with the reasons. This holds for all groups. We compute the significance with a Mann-Whitney-U test and find \( p < 0.001 \) for all comparisons.

To find out what this means, we investigate the difference in reading behaviour of the users in the three groups before, during and after the experiment. We plot all these differences in Figure 8.12. We find that the users that look at reasons during the experiment also read more than users that do not look at reasons during the experiment when we measure their reading behaviour before and after the experiment. (Again we compute the significance scores with a Mann-Whitney-U test and find \( p < 0.001 \) for all comparisons.) Therefore, the difference we observe is more likely to be a correlation than a causation, i.e., users that are more active in general, also look at reasons more often. We see that the difference is less during the time the experiment was live. We ran a paired t-test,
Table 8.6: Feature reasons and rest reasons that are used, numbered.

<table>
<thead>
<tr>
<th>Number</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature 0</td>
<td>item rating score</td>
</tr>
<tr>
<td>Feature 1</td>
<td>item pick probability</td>
</tr>
<tr>
<td>Feature 2</td>
<td>item number of images</td>
</tr>
<tr>
<td>Feature 3</td>
<td>item channel followed by user</td>
</tr>
<tr>
<td>Feature 4</td>
<td>item provider followed by user</td>
</tr>
<tr>
<td>Feature 5</td>
<td>newsletter purchased channel score</td>
</tr>
<tr>
<td>Feature 6</td>
<td>newsletter purchased provider score</td>
</tr>
<tr>
<td>Feature 7</td>
<td>user item negative channel feedback</td>
</tr>
<tr>
<td>Feature 8</td>
<td>user item negative provider feedback</td>
</tr>
<tr>
<td>Reason 9</td>
<td>diversifier 1</td>
</tr>
<tr>
<td>Reason 10</td>
<td>diversifier 2</td>
</tr>
<tr>
<td>Reason 11</td>
<td>must read</td>
</tr>
</tbody>
</table>

Table 8.7: Heuristic reasons, numbered.

<table>
<thead>
<tr>
<th>Number</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reason 0</td>
<td>Because you often read about CHANNEL.</td>
</tr>
<tr>
<td>Reason 1</td>
<td>Because often read from PROVIDER.</td>
</tr>
<tr>
<td>Reason 2</td>
<td>Because often read from AUTHOR.</td>
</tr>
<tr>
<td>Reason 3</td>
<td>Because you are interested in CHANNEL.</td>
</tr>
<tr>
<td>Reason 4</td>
<td>Because we think CHANNEL could be interesting for you.</td>
</tr>
<tr>
<td>Reason 5</td>
<td>Because we think PROVIDER could be interesting for you.</td>
</tr>
<tr>
<td>Reason 6</td>
<td>Because you follow CHANNEL.</td>
</tr>
<tr>
<td>Reason 7</td>
<td>Because you follow PROVIDER.</td>
</tr>
<tr>
<td>Reason 8</td>
<td>Because you seem to like a long read every now and then.</td>
</tr>
<tr>
<td>Reason 9</td>
<td>The editors really liked this piece.</td>
</tr>
<tr>
<td>Reason 10</td>
<td>According to the editors, this is one of the best stories of the day. No matter your preferences.</td>
</tr>
</tbody>
</table>
where we counted reads per day and matched on days and found $p < 0.025$ in all groups. (Using a Bonferonni correction, we use $\alpha = 0.05/2 = 0.025$ as significance level.) It is difficult to point at a potential reason for this. The fact that the A/B-test was live is not the only difference between the three time periods. We also need to take factors like summer holidays and general differences in reading behaviour over time into account. However, the trend that users that interact with reasons during the time the experiment was live read more than the other users, is a very clear trend in all groups, in all periods of time.

In the next chapter we summarize our findings and conclude our research. Moreover, we give directions for future research.
Chapter 9

Discussion and conclusion

In this study we have focussed on the explainability of rankings. We have distinguished between two explanation audiences: the users of a system and the developers of a system. In our design, we have mainly focussed on this first group. We have used the recommender system of Blendle as our problem environment. Throughout this study we aimed to answer five research questions. The first two research questions looked into whether the users of an online news recommender would like to receive explanations for the recommendations they received and what these explanations could look like. The last three research questions focussed on the design of a model to explain rankings. We aimed to make explanations that faithfully describe the underlying structure of the ranking algorithm. Moreover, we wanted our design to be model agnostic, and able to generate explanations in real time, so that it can be used in a production environment. To this end we proposed LISTEN and Q-LISTEN.

In the remainder of this chapter we summarize our findings on our research questions, we look into the theoretical and practical implications of this work, restrictions of the setup and directions for future work.

9.1 Answers to research questions

In this study we aimed to answer five research questions. In this section we summarize our findings on each of these questions.

**RQ 1 - Do users want to receive explanations of why particular news items are recommended to them?** Yes, they do. We sent out a questionnaire to 541 Blendle users and asked them this question. 120 users returned the questionnaire and the large majority of this group indicates that they would like to receive explanations for their recommendations.

**RQ 2 - What way of showing news recommendations do users prefer?** From the same questionnaire as we used to answer our first research question we learn that users do not have a clear preference for a particular way of how news recommendations are shown to them. We did find that users ‘either loved or
hated’ a bar chart reason type, where reasons are presented in the form of filled bars. Therefore we recommend to stick to reason types that are more neutral, so that the reasons are understandable for everyone.

RQ 3 - How do we provide users with easy to understand, uncluttered, listwise explanations? We decided to develop a method that, at first glance, is indistinguishable from a method that makes pointwise explanations. I.e. reasons are still given in a textual form, such as ‘You see this article because you often read from provider X’. However, the reasons that are computed by our listwise method are more faithful than the reasons that are computed by a pointwise method, as we do compute the importance scores of the features of an item by looking at their effect on the entire ranking.

RQ 4 - How do we build an explanation system that produces faithful explanations for the outcome of a ranking algorithm, yet is scalable so that it can run in real time? The answer to this research question comes in two parts. First we proposed LISTEN, to construct faithful explanations. LISTEN computes the impact of each feature in the ranking. The impact of a feature is computed by calculating how much the ranking changes if the value of this feature is changed. For this, we sample possible feature values and for each new value we compute the Average Precision Ranking Correlation (Yilmaz et al., 2008). Features that are found to be most important by this metric are returned as explanation for the item.

This method is not scalable. The sampling of the feature values and computing the effect on the ranking is too costly to run in real time. Therefore we increase the computation speed in two ways. First, we divide the method that we described above in two parts: a training part and an explanation part. In the training step we make so called disruptive distributions. These distributions are found by following the method described above and are made to find the most disruptive feature values for a feature. We only use these feature values to compute reasons in the explaining step. This decreases the number of feature values one needs to try to find the most important features. As a second speed up step we propose Q-LISTEN: we train a neural network on the data that we have found in the explaining step. Especially this decreases the computation speed drastically and ensures that the model can run in production. In production the neural network produces the reasons.

RQ 5 - Does the reading behaviour of users who are provided with model-agnostic listwise explanations for a personalized ranked selection of news articles differ from the reading behaviour of users who are provided with heuristic or pointwise explanations for a personalized ranked selection of news articles? We used an offline and an online evaluation to answer this research question. In the offline evaluation we used dummy data to show that LISTEN produces faithful reasons and indeed uses the impact of a feature on the entire ranking instead of just its impact on the score of the specific item. The addition of the disruptive distribution and the use of neural networks did slightly decrease the accuracy of the approach. However, the addition of both steps ensures that we can produce reasons on the fly in production and this makes it worth to allow a slight decrease in accuracy.
An online A/B-test on all Blendle users that receive a personalized news selection revealed that users did not change their reading behaviour based on one of the reason types. In all groups users did not show much interaction with the reasons that were available.

9.2 Theoretical and practical implications

In this research we have seen that users want explanations for their recommendations. Yet when it comes down to the interaction with these explanations we have not observed any differences for different types of reasons. Users were not conclusive as to which design they would prefer, they did not interact differently when different reason types were shown, nor did they interact a lot with reasons at all. This may raise the question whether we should put such a lot of effort in designing (faithful) explanations. However, given the fact that users state they would like to receive explanations, it is recommended to accept this and design faithful reasons. Even though users may not notice the difference or may not respond differently to them, it is important to provide them with truthful information, as in the end only true reasons answer the user’s question “why do I see this?”. Moreover, not only do users ask for explanations, the GDPR also enforces to be able to explain algorithmic decisions. LISTEN and Q-LISTEN can be used to faithfully explain the decisions of any ranking algorithm, in real time. The method can still be improved on and we go into that in the next part of this chapter.

9.3 Limitations and future work

In this section we look at some limitations of the current setup and directions for future work.

Decreasing the amount of apriori information

In the current approach we request apriori information of the feature types we deal with: we want to know whether a feature is continuous, discrete or predefined. In this, we follow the approach that was also used in Ribeiro et al. (2016). Ideally, we would not need this kind of information. We do add this information for two reasons. First, the computation time of the model increases when one needs to try less feature values. Secondly, it is unintuitive to try feature values that do not exist in the data and let decisions depend on that.

One could extract this type of information directly from the data. This would require an additional step, in which a check is built in to find out what feature type is encountered. For example, if the values for a particular feature contain integers only, this feature could be labeled as discrete. If only a few feature values are encountered, this feature could be assumed to be of the predefined type. A hyper parameter can decide on the precise value of ‘a few’. If all sorts of feature values are found for a feature, the continuous label can be assigned.
Dealing with diversification

In the current setup we have chosen to leave the diversification out of the explaining pipeline as, technically, the diversification step does not involve a ranking problem. However, we could include the diversification into the current pipeline, with a few modifications. We describe these modifications in this section.

The diversification pipeline is an additional step of the algorithm to provide users with a personalized selection of articles. The diversification is done based on a number of characteristics of a single article and the articles that are already ranked. We could make features such as ‘this article is from provider X’ and add these features to the current feature space. Then we can proceed with the current pipeline. We could change the feature values of the new features as well, to find out how disruptive they are. We can make training data for our neural network and train the networks on these new data points. Adding these features increases the feature space, as we need features for every single provider and for every single channel. Potentially this decreases the ability of the network to learn the underlying structure of the data. In Section 9.3 we look into precisely this issue.

Dealing with less concrete features and trying more rankers

For the current application, we have dealt with very concrete features that we could easily translate to the user. However, in many applications nowadays, these features are less concrete as features are directly extracted from the raw data instead of manually designed. The technical setup would still work and the most important features could still be found. Yet the question is how these features should be communicated to the user. For translation tasks or image classification tasks this is not as difficult, as for example Hendricks et al. (2016) show for images. For translation tasks one could use the attention module (Bahdanau et al., 2014). For ranking this is still an open issue, that is important to address in future work, as the field of deep learning for ranking starts to become more popular these days (e.g. Seo et al., 2017; Serrà and Karatzoglou, 2017; Severyn and Moschitti, 2015), which inevitably causes the interpretability of feature values to decrease.

Moreover, in this research we were restricted to Blendle’s data and the ranking function used by Blendle on the one hand, and the ranking function and toy data set that we have composed ourselves on the other hand. In order to be able to strengthen the claim that LISTEN is model-agnostic and can be used for any type of ranking function, it would be good to test it on more rankers and on more data sets in future research.

Neural networks to learn the mapping between input and explanations

For our current application the neural networks were very well able to learn the underlying structure of the data. There are a few points to take into consideration when we extend this work to other domains. In this section we look into these points and give suggestions how to approach these issues in future work.
Structured input data  The input data that we train the neural network on is quite structured. Many users behave similarly and therefore feature values take on similar values, even though in theory they could take on an infinite number of values. Presumably, when the input data is less structured, it is more difficult for a network to find the underlying structure of the input to output mapping.

Dealing with input of variable length  Currently we bound the input size of the model. Ideally we would want this input size to be variable. This ensures that if, for some reason, there are less than the $N$ items in a user’s ranking, the network would still work. Now, this barely happens, but we do need to have a fall back in case it would. In general, it is good to make the model as flexible to external changes as possible.

Suggestions to solve the above issues  For the listwise method, both issues could be solved by using sequence-to-sequence learning with Recurrent Neural Networks, RNNs (Sutskever et al., 2014). The listwise approach would be a good fit for a sequence-to-sequence approach, as we want to learn importance based on concurrent structures. Every single feature vector in the ranking would serve as an input block for the RNN. We would need to use a bidirectional approach (Schuster and Paliwal, 1997) as we want to predict the importance of the feature based on items before the current item as well as on items after the current item. The potential downside of using an RNN is that we want to predict importance based on all elements in the sequence and not only based on elements that occurred just before or just after the current input. This could be solved by using attention (Bahdanau et al., 2014).

RNNs are well able to work with input data of variable length, due to their recurrent structure. Also the fact that we feed the input vectors one by one, resulting in a single summarization vector, makes it possibly easier to deal with less structured data.

The reason that we have not used a sequence-to-sequence approach in the current research is that we found our multilayer perceptrons to learn the underlying structure of the data to a very high accuracy. It is recommended to try the sequence-to-sequence approach in future work when different input data is used.

Training predictions and explanations hand in hand

In this research we have made an explanation model that can be used on top of another ranking model. In the ideal situation a model would be able to explain itself. In Chapter 2 we have described a few of these approaches. By nature, these approaches cannot be model-agnostic in the interpretation that we have used in this research. However, one could think of approaches that can be ‘added’ to current models and that are trained together with the rest of the model.

For future work it is recommended to directly integrate an explainability component to a freshly designed machine learning algorithm. This would improve the explainability of machine learning algorithms in general and therefore a very valuable addition.
Bibliography


Appendix A

User study
Table A.1: Mean and standard deviations of the scores on different types of judgments in the user study. The “reason types” refer back to the types of reason listed in Table 5.1.

<table>
<thead>
<tr>
<th>Reason type</th>
<th>Question</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Std</td>
<td>Mean</td>
<td>Std</td>
<td>Mean</td>
<td>Std</td>
</tr>
<tr>
<td>Sufficiency</td>
<td>3.530</td>
<td>1.076</td>
<td>3.625</td>
<td>1.028</td>
<td>3.333</td>
<td>1.098</td>
</tr>
<tr>
<td>Trust</td>
<td>3.000</td>
<td>1.115</td>
<td>3.250</td>
<td>1.122</td>
<td>3.032</td>
<td>1.023</td>
</tr>
<tr>
<td>Satisfaction</td>
<td>3.606</td>
<td>0.919</td>
<td><strong>3.661</strong></td>
<td>0.969</td>
<td>3.317</td>
<td>1.096</td>
</tr>
<tr>
<td>Average</td>
<td>3.458</td>
<td>0.798</td>
<td>3.580</td>
<td>0.836</td>
<td>3.317</td>
<td>0.916</td>
</tr>
<tr>
<td>Type</td>
<td>Transparency</td>
<td>Sufficiency</td>
<td>Trust</td>
<td>Satisfaction</td>
<td>Average</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
<td>-------------</td>
<td>-------</td>
<td>--------------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>Type 2</td>
<td>U = 1811.0, p &gt; 0.05</td>
<td>U = 2059.0, p &gt; 0.05</td>
<td>U = 1597.0, p &gt; 0.05</td>
<td>U = 3858.0, p &gt; 0.05</td>
<td>U = 1684.0, p &gt; 0.05</td>
<td></td>
</tr>
<tr>
<td>Type 3</td>
<td>U = 1868.0, p &gt; 0.05</td>
<td>U = 2147.5, p &gt; 0.05</td>
<td>U = 1571.5, p &gt; 0.05</td>
<td>U = 4005.0, p &gt; 0.05</td>
<td>U = 2038.5, p &gt; 0.05</td>
<td></td>
</tr>
<tr>
<td>Type 4</td>
<td>U = 1860.5, p &gt; 0.05</td>
<td>U = 2016.5, p &gt; 0.05</td>
<td>U = 1512.0, p &gt; 0.05</td>
<td>U = 3001.0, p &gt; 0.05</td>
<td>U = 1938.0, p &gt; 0.05</td>
<td></td>
</tr>
<tr>
<td>Type 5</td>
<td>U = 1422.5, p &gt; 0.05</td>
<td>U = 1397.0, p &gt; 0.05</td>
<td>U = 2900.0, p &gt; 0.05</td>
<td>U = 1441.0, p &gt; 0.05</td>
<td>U = 1369.5, p &gt; 0.05</td>
<td></td>
</tr>
</tbody>
</table>

Table A.2: Statistical differences between reason types, between different questions.
Table A.3: Sample sizes per reason type

<table>
<thead>
<tr>
<th>Type 1</th>
<th>Type 2</th>
<th>Type 3</th>
<th>Type 4</th>
<th>Type 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>66</td>
<td>56</td>
<td>63</td>
<td>55</td>
<td>120</td>
</tr>
</tbody>
</table>