Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Erik Bjørgen
Carl Johan Hambro

Explaining deep learning methods for recommender systems

Master's thesis in Computer Science
Supervisor: Heri Ramampiaro
June 2019

NTNU
Norwegian University of
Science and Technology

Abstract

The requirements for recommendation systems have changed. How recommendations are presented to the users, as well as GDPR, has brought new demands to the recommendations. Explainability is a new requirement which is hard to find in high-performance black box models. We discuss relevant theory and approaches for current explainability techniques. Then, we attempt to increase the explainability of a deep learning-based recommender system by using said theory. This includes inspecting the model by examining its input and by looking at what the model has learned with a global method. Finally, we validate the insights gained by removing input data and by using statistical tools. The former method measures the performance loss when removing the input, which is related to the explanation. Finally, we summarize our findings of explainable recommender systems and propose avenues for future work.

Table of Contents

Abbreviations

CNN Convolutional Neural Network

DARPA Defense Advanced Research Projects Agency

HCI Human–computer interaction

LRP Layerwise Relevance Propagation

MF Matrix Factorization

ML Machine Learning

NARRE Neural Attentional Rating Regression with Review level Explanations

NCF Neural Collaborative Filtering

NLP Natural Language Processing

SOTA State Of The Art

TCAV Testing with Concept Activation Vectors

XAI eXplainable Artificial Intelligence

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Recommendation algorithms hold the power of relevant and accurate predictions. These prediction-based systems provide personalization to users, be it either groceries, songs, movies, articles, clothes, or even other users in dating platforms.

Provided a set of users and their preferences for items, the system should predict the next top-N items for a given user. Naturally, business metrics related to customer satisfaction or customer retention often depends on the accuracy of these recommendations. The intense focus on performance metrics alone has left other aspects less charted, e.g., privacy concerns and ethics. GDPR came into force recently, and one of the aspects it covers is the *what*, *why*, and *how* concerning the data gathered. Users must be able to see *what* data is collected by the service. This data can be explicit actions such as ratings, reviews, or likes. It can also be implicit actions such as page views, clicks, mouse hovering, time spent on certain parts of the web page, and other interactions. The use of this data must also be transparent, i.e., the purpose or the *why* of the data gathering. Finally, the *how* is covered by providing insight into the data presented to the user. In our case, this would be insight into the predictions that are given to the end user.

Another driving force for explainability besides ethics and laws are the users; human-computer interaction also has a role in user satisfaction. There is typically a lack of communication between the system itself and the end user. Any recommendations can be conceived as not trustworthy as there is no basis provided for the decision. I.e., the end user cannot verify the validity of the prediction. This trust is weakened more when an incorrect prediction is given. This makes explainability a vital aspect of recommender systems. As a bonus, explainability can also be used to confirm the inner workings of a model and its assumptions. It can also uncover issues such as biases.

The *how* is a difficult task, as many algorithms are considered a black box. Latent Factor Models [3] for instance, have been integrated into the idea of collaborative filtering to improve rating predictions. It is, however, difficult to understand the predictions from such systems since the latent factor models do not possess inherent meanings. Methods based

on deep learning have similar issues. In short, these black boxes can provide excellent recommendations, but at the cost of not providing any knowledge about the problem at hand. One issue with most machine learning approaches, and with deep learning, in particular, is that the models only learn the dataset it has been trained on, which may not necessarily be representative of the real world. This complicates the deciphering of the inner workings further. However, the scientific community has recognized the lack of explainability as an issue, and there is a slow but steady trend to address this issue. DARPA uses the term XAI, which is often used for this research area, but *explainability* and *interpretability* is often used interchangeably.

White box models do provide an interpretable model, but these are often simpler and performs worse than the complex black box models. Linear regression is a typical example of this. Even if the model is interpretable, it might not handily provide a layman explanation of the prediction which should be clear, concise, and human-readable.

This brings us to another issue: evaluating the explanations is a hard task as there is no basis or ground truth of which one can benchmark against. A labeled dataset would be domain specific and limited to its scope. A movie recommender system with one explanation component would need a dataset based on that exact system. There have been some attempts at offline metrics, but online testing or case studies remains the current standard. It should be noted that this applies when an explanation is defined as a usercentric justification and not a simple input-bound explanation. Some look for explanations by inspecting the input, leaving much of the interpretation to humans. This might be a suitable method for providing the *how* in layman terms since the input is usually in familiar terms to the user. Others inspect what the model has learned in general, and provide this as the interpretation of the model. The model prediction explanations are based on these interpretations. Such methods might appeal to technical stakeholders but might not fare well with end users as they are often technical. There are other approaches as well, but these two are enough to show that different stakeholders perceive interpretability and explainability different. During our work, it became apparent that there is no clear consensus on the definitions of *interpretability* and *explainability*. These words cover a wide range of approaches within AI and ML.

1.2 Problem description

How can we make methods used in black box recommender systems explainable?

Q1 How do current recommender systems provide explanations for their recommendations, and what limits are imposed on such explanations?

Q2 What techniques exist for explaining deep learning models?

Q3 How can different parts of deep learning-based recommender systems be explained? How can insight into these components provide better explanations?

Q4 How can we validate explanations?

1.3 Scope and limitations

We discuss recommender systems in general, including white box models, and highlight relevant features of these systems. RQ2-RQ4 is narrowed by inspecting a single recommender system: NARRE [4]. NARRE uses user reviews to estimate user ratings for items and provides a simple explanation for these ratings by supplying the most contributing review along with the recommendations. We believe that such explanations leave much to be desired for the user, which makes the explanation challenging to interpret; the opposite of how an explanation should be. Our approaches vary from global explanations to local explanations, by combining the model with recent advances in XAI, and by inspecting the input data with the help of clustering.

1.4 Thesis structure

Chapter 2 provides theory about classical recommender systems, evaluation methods, deep learning, machine learning concepts, explainable AI, and finally, the combination of these topics. A survey of related work is presented in section 3.1, where different works are summarized and discussed regarding explainable recommender systems. Section 3.2 describes different datasets, which were available at the time. We inspect a deep learning-based recommender system with different approaches in section 4 along with a technical description in section 4.2. The results from our approach and the evaluations of these are in chapter 5 with a discussion before chapter 6 concludes the results along with future work.

The theory in chapter 2 partially covers Q1 to Q4. We choose to inspect a single model, with the results of this in section 5, which then answers Q3 and Q4 more in-depth.

Chapter 2

Background

Recommender systems are within the field of *machine learning* as *supervised algorithms*. They are supervised because the data is said to be annotated or classified. A set of input parameters or features are provided with an annotation. As an example, a general recommender system could have user interactions as two input parameters, user ID and item ID, while the response variable, the annotated value, is the degree of likeness. Some systems focus on returning the top-N items to recommend, rather than having high accuracy for all users and all items. The trade-off in overall accuracy is offset by the importance of the recall of the top-N items.

Classical recommender systems are presented in section 2.1. Section 2.2 introduces general theory for deep learning as well as some theory for natural language processing. A mix of machine learning theory is gathered in section 2.3: different evaluation methods and metrics, desiderata of recommendations, the phenomenon *curse of dimensionality*, *out-of-vocabulary* for word embeddings, and clustering. Section 2.4 introduces explanations and interpretations of machine learning models, while section 2.4.2 dives into XAI for deep learning-based models. Finally, section 2.5 is a combination of recommender systems, deep learning, and explainable AI by introducing an algorithm called NARRE.

Feedback types Recommender systems consider user feedback as either explicit or implicit. The former are explicit actions by the user, such as ratings on items: thumbs up or down, or a scale from 1 to 10.

This feedback type can be problematic in practice since it is challenging to get users to interact with the system actively. This is not a problem with implicit ratings, where the behavior of the user is used to infer the preferences of the user. E.g., a user that frequently visits the same item detail page is likely interested in that item. A drawback of implicit actions is the difficulty in distinguishing between aversion for the item and indifference.

Some platforms have reviews written by users. These reviews have the potential to express a more nuanced view of an item from the user, which could improve the predictions. Hao et al. [5] show that other types of explicit feedback from users are noisy and that using a model which includes the reviews to predict ratings can lead to better predictions.

Cold Start and The Long Tail Cold start is the term for a problem affecting both users and items in a recommendation system.

In the context of users, it refers to the situation where a new user has not given any feedback to the system. Recommender systems, in general, have trouble providing better recommendations than random recommendations to these new users. A standard solution is to recommend relatively popular items, in the hopes that since many people enjoy these items, a random user will like it too. In the domain of movie recommendations, one can also recommend new and popular movies, since these might be the reason the user joined the platform.

The problem is similar in the context of items. When a new item is added, the system has insufficient data about the item to recommend it to users. While the cold-start problem for users is a common problem in several domains, this problem affects some systems more than others.

Related to this problem is the concept of the long tail. Movie revenue distribution, for instance, follows a power law, where a few blockbusters represent a relatively large portion of the revenue. These popular movies are generally safe to recommend to both existing and new users. As a result, popular movies remain popular, while less popular movies will rarely be recommended. This leads to a paradoxical situation where niche movies are never recommended because they are not popular but needs more popularity to be recommended. The system will never become more confident as to when a niche movie should be recommended. This group of less popular movies, or niche movies, is called the long tail. Following the power law, the long tail amounts to a more significant portion of the total revenue than the few blockbusters. This applies to many domains in recommender systems.

2.1 Classical recommender systems

There are different approaches to providing top-N predictions to users based on explicit or implicit feedback. The two main approaches are Content-based recommendations and Collaborative filtering.

2.1.1 Content-based

In Content-based Recommendation, the predictions are based on the properties of the content the user previously has expressed interest in. In the domain of movies, such properties can be actors, directors, movie genre, tags, language, and so on. The goal of the system is to find a movie which is as similar as possible to the movies previously liked by the user. Formally it is based on a prediction heuristic, $sim(d_i, d_j)$, which measures the similarity between two feature vectors. This is used to rank all items against a single user. This similarity measure between two vectors is often done with cosine distance or Pearson coefficient, the former shown in equation 2.1.

$$sim(d_i, d_j) = \frac{\sum_k w_{ki} \cdot w_{kj}}{\sqrt{\sum_k w_{ki}} \cdot \sqrt{\sum_k w_{kj}}} \tag{2.1}$$

This user-user independence makes content-based systems less prone to issues with recommending movies from the long tail. It can be seen as personalized, fitting to each user's specific tastes. The drawback is overspecialization of the user profile. The recommendations will not be novel, but remain safe within proximity of the user profile vector, i.e., the *serendipity* remains low. This metric is discussed in section 2.1.3.

It does not suffer from the cold start issue on items since it does not depend on other users' ratings. The feature vector for an item can be constructed from the metadata. Such feature vectors make it possible to recommend both new movies and niche movies. New users still pose an issue as they do not have any ratings as a basis for their user profile. The user-user independence is a double-edged sword: It is not dependent on other users' ratings, but it cannot use the crowd knowledge to its advantage.

Finally, explanations are built into the user profile where each feature has a contributing weight to each recommendation. It is simple to extract the *why* of a recommendation in such systems as the feature weights are easy to interpret. Another advantage is the ability to correct the user profile quickly. Tweaking the feature weights would be straightforward and intuitive, trading accuracy for convenience. A drawback of explanations that are based on a comparison of the feature vectors of movies is that it is difficult to get more detailed and nuanced information from the system. The same actor may play different roles in different movies, and the performance and synergy with other elements of the movie will vary.

2.1.2 Collaborative Filtering

Crowd knowledge is used in collaborative filtering. The predictions are based on implicit or rating-based latent relationships between users and items. The implementation is not domain specific as there is no need for feature engineering, unlike content-based methods. It does suffer from the cold start problem for both users and items since it relies on feedback. It is also biased towards more popular items, reducing the diversity of the ratings [6].

Both users and movies have a bias. Popular movies tend to have a higher mean rating, and users also have an individual inherent rating bias. The distributions of the ratings are normalized by subtracting the bias. This process is performed at the item and user level, as shown in equation 2.5. Two main models are commonly used in collaborative filtering; these are discussed below.

Neighborhood models These models can be user-centered (user-user) or item-centered (item-item). In the user-centered case, the rating of an item for a user is based on similar users' explicit rating of that item. Finding similar users is done on the assumption that if two users have similar ratings on some items, they will have somewhat similar tastes. The estimated rating for an item and a single user is ideally done by considering a large number of other users. This increases the robustness of the rankings, making it less sensitive to small changes in the data. The item-centered case is similar. Predicting the missing rating, in this case, is based on the users rating of similar items. The similarity between items for the user is based on the similarity of the ratings made by the user on the items. In an

intuitive sense, ratings cluster similar items together.

$$\hat{r}_{ui} = \mu_i + \frac{\sum\limits_{j \in N_u^k(i)} \text{sim}(i,j) \cdot (r_{uj} - \mu_j)}{\sum\limits_{j \in N_u^k(i)} \text{sim}(i,j)} \tag{2.2}$$

Equation 2.2 is a k-NN inspired approximation of the rating of item i on a user. μ_i is the mean of all ratings for item i and μ_u is the mean of all ratings from user u. $N_u^k(i)$ is the k nearest neighbors of user u which has rated the item i. Explanations are provided by listing the subset of items or users which are known to the user, sorted by distance from the recommended item. A typical example would be *You might like item-3 since you liked item-1 and item-2*. This explanation can be problematic to rely on since it is not a complete picture of everything the model based its prediction on. Even if the user had not liked item-1 and item-2 in the example above, the system might still recommend item-3 since it bases the recommendation on a potentially large number of other items.

Latent factor models These models are similar to the content-based approach. They attempt to model each item along many different dimensions that are, in a sense semantically clustered factors. The difference from content-based is that the factors are not determined by human domain experts, but rather by the algorithm itself. This means that the factors could coincide with some characteristic of the item that is used in a content-based approach, but the factors would typically be complicated for humans to interpret. This interpretation would be similar to manually identifying clusters in neighborhood models and interpreting the representations of these. A linear approximation can be done to express how these hidden factors interact to recommend an item for a single user.

The perceived interests of the user are also mapped to this latent space, and the items are ranked based on their distance to the user interests. These factors are sometimes referred to as topics, mainly when they are used to interpret the inner workings of recommendation systems.

Let A be the user-item rating matrix. In this method we assume that A can be decomposed into item-factor matrix Q and user-factor matrix P, such that $A = Q \cdot P^T$. The dimensions of Q and P are dependent on the number of latent factors and are usually less than the original A matrix. A variant of this matrix factorization is called Singular Value Decomposition, SVD, with its general form in equation 2.3.

$$A = U\Sigma V^T \tag{2.3}$$

The estimated rating $\hat{r}_{ui}$ for a user u and item i is given in equation 2.4 and is a practical extension of SVD [7]. μ is the average rating over all movies. b_u and b_i is the user u and item i bias, respectively. This bias is the difference between the users' average rating and the average of all ratings. Similarly for b_i, which is the difference between the average rating of a movie compared to the average rating of all movies. q_i is a vector that captures the weights for the latent factors for a given item i. Likewise for p_u, but for a given user u. In equation 2.3, q_i corresponds to the the ith row of U and p_u is the uth column of the product ΣV^T, often called P.

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u \tag{2.4}$$

SVD gained much attention [8] since it reduces the dimensionality of the problem. But, it can not be used directly because of missing values or ratings in A. An approximation is used instead. This turns the problem into a task of optimization with the help of Stochastic Gradient Descent or Alternating Least Squares. The optimization problem in the context of recommendations is shown in equation 2.5. r_{ui} and $\hat{r}_{ui}$ is respectively the true rating and the estimated rating of user u for item i. λ is a regularization constant which aims to avoid overfitting by adding a cost to the sizes of the latent factors. It uses the L2-norm, which promotes sparsification and therefore reduces the number of latent factors.

$$\min_{q,p} \sum_{r_{ui} \in R_{train}} \left(r_{ui} - \hat{r}_{ui}\right)^2 + \lambda \left(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2\right) \tag{2.5}$$

A variant of this equation [9] makes it possible to decompose the user factorization, p_u, as a linear combination of the interactions of user i. This is an important detail as it allows us to create explanations from the latent space. These explanations would be on the same form as the neighborhood-based models since the user feedback composes the model. However, the true meaning behind the latent factors would still be unavailable.

2.1.3 Desiderata of recommendations

This subsection introduces the desiderata of recommendations encountered in the literature.

Novelty Novelty depends on, as implied by the name, novel items in the recommendations. The implicit action of a user not choosing a specific recommended item could imply that the recommendation is inaccurate. However, since the user in many contexts only can choose one item, this metric is not entirely accurate. To find a balance, it is possible to measure the novelty of the recommended items and use that as part of the evaluation metric for each item. These metrics can often be classified into different levels depending on the granularity of the novelty. Kapoor, Kumar, Terveen et al. [10] require novel movies to be previously unknown by the user. This level of novelty can, however, be hard to measure by the system, since the information required is outside of the context of the system. Requiring the item to be utterly unknown to the user is sometimes [11] relaxed so that a novel movie is one which the user has little knowledge of before the recommendation, but not that it is entirely unknown. In practice, the degree of how much an item is known to a user will often depend on the level of interaction between the user and item.

Some authors [12] define novelty as items that have never been recommended to a user at all, regardless of whether the user has interacted with the item or not. Recommender systems that consider the novelty of an item must as mentioned, define the level of interaction to consider an item to be known to a user. Besides, it can often be useful to consider the pairwise similarity between the items so the items recommended are not too similar. Allowing the novelty of one item to be affected by the novelty of another depending on their degree of similarity is closely related to the granularity of the chosen definition of novelty in the system.

A related concept is the desideratum of *diversity*. Ziegler et al. [13] defines it as the similarity between two lists of recommended items. They argue that it is crucial that the recommendations a user receives from a system represent a diverse set of topics the user finds attractive. This metric is also dependent on a distance measure between each item.

Informativeness Informativeness [14] in the context of supervised learning describes how much the model explains its output. This measure is closely related to the degree to which the system is explainable. Kim et al. [15], and Huysmans et al. [16] describes a setting where the system provides information to the user, and the user makes the decision. The real task of this system is to provide useful information to the user and to aid the user in exploring the data, instead of predicting ratings or clicks. It is the opinion of the authors that this term has been replaced in the literature by the broader acronym XAI. However, the importance of the concept is still emphasized by researchers.

Serendipity and Unexpectedness Unexpectedness is the idea that users prefer to be recommended unexpected items instead of the recommendations that are obvious to the user. Since the set of expected items for each user is difficult to determine, the items recommended by a primitive recommender is sometimes used as an approximation to this set [11, 17]. The unexpectedness of a recommender system can then be defined as the rate of disagreement with the primitive recommender.

Defining a measure for the unexpectedness of an item is also possible without relying on a primitive recommender. Bridge and Kaminskas [18] uses the Pointwise Mutual Information (PMI) measure [19] to define the unexpectedness of two lists of recommended items. The PMI of two items is described in equation 2.6. $p(i)$ is the probability that item i is rated by a user, and $p(i, j)$ is the probability that a user has rated both item i and item j.

$$PMI(i, j) = ln(\frac{p(i, j)}{p(i) * p(j)}) \tag{2.6}$$

Bridge and Kaminskas [18] then defines the unexpectedness of an item in relation to a user to be either the maximum or the average PMI between the item and all the items in the user profile. Since the probability that a user rates an item can be inferred from the rating matrix, this measure can be considered more stable than the version that depended on the primitive recommender.

A critique of unexpectedness is that a system based on maximizing the unexpectedness of recommended items might be unreasonably biased towards recommending rare items. A desideratum that expands on unexpectedness is serendipity. Serendipity is defined in Herlocker et al. [20] as "a surprisingly interesting item he might not have otherwise discovered". Measuring how interesting the user considers the recommended item can be a vague term and is often approximated as the expected utility or rating of the item. Several contributions to the literature [11, 12, 17] describe serendipity as dependent on being unexpected, and uses a measure of unexpectedness similar to those described earlier in this section multiplied by the utility of the item as a measure of serendipity. Using a definition of serendipity which does not rely on the expected utility of an item can be advantageous to the system, an idea noted and expanded upon in Herlocker et al. [20]. The system can then be designed to widen the interest of its users. More suitable evaluation metrics for

such systems could be Click-through rate and View-through rate, as defined in equations 2.10 and 2.11.

Trust Users decide to watch a movie based on the information they have about it, whether that is word-of-mouth, trailers, or even movie posters. In order for users to choose to base their decision on the recommender system, they need to have a certain amount of trust in the system. A welcome side effect of explainable recommender systems is that it increases the trust users have in the system. For recommender systems without explanations, the only offline measure of trust is the accuracy of the system. Alternative measures for trust could be a survey asking the users about their perceived credibility of the system. A high return rate of users could also be indicative of trust. Some of the previously mentioned desiderata could have a short-term detrimental effect on trust. If the system attempts to maximize the serendipity of its users, it might place an unreasonably high value on unexpected movies which could increase the chance of hitting the jackpot at the expense of a lower average utility. A strategy, which undermines serendipity, is to recommend items that the user already have seen and rated high. The recommendation will not be useful to the user, but since the user can recognize it as a high-rated movie, the system might seem more trustworthy.

2.2 Deep learning

The theory in this section is compiled from [21] and [22].

Considering a supervised learning problem, with input vector $\boldsymbol{x}$ and target y, the network or model is to create a prediction $\tilde{y}$ which matches y as well as possible. The distance or loss is often measured using RMSE or MAE for regression problems and binary cross entropy for classification problems. See section 2.3.1 for details on the RMSE and MAE metrics. The network consists of layers, where each layer is a set of nodes or neurons. Each node receives input from the previous layer and passes its output to one or several nodes in the downstream layer. A node performs some operation, such as addition or multiplication, usually with the input and an internal weight. An addition layer with internal weights is often referred to as a *bias layer*.

In classification, the last layer produces a vector of probabilities for each of the target classes. These probabilities must lie between 0 and 1, hence a *sigmoid* is often used. The sigmoid is one of several *activation functions*, which are functions that determine the output of a node. Another popular activation is the *ReLU*. It is commonly used on hidden nodes in the network, as it does not lead to vanishing gradients as the sigmoid does. The ReLU can also be used as the output node in a regression setting. These are the main building blocks of a simple neural network.

The process of passing input to the head and propagating the data throughout the network is called the *feed forward phase*. The weights of the network are trained by *back propagation*. This involves partially differentiating the loss function with respect to all the weights in all layers. The output of the last layer is partially differentiated w.r.t. the input from the previous layer. This is repeated until the input values, $\boldsymbol{x}$ are reached. The change in weights, ΔW, for all layers are multiplied with the learning rate and finally added to

the original weights, thus updating them. The feed forward phase and back propagation phase are repeated for every case in the data set.

Overfitting is a problem in neural networks due to the number of parameters that are tuned. Overfitting occurs when the model has higher accuracy on the dataset than any other representative set of the domain. This is described further in section 2.3.1. The goal of training a model is to generalize it to the domain, so this behavior is undesired. There are several *regularization techniques* to cope with overfitting. *Dropout* [23] is the most notable in our study. A specified percentage of the nodes in a selected layer are clamped at 0, i.e., some nodes are disabled at random for each forward pass. None of the nodes are clamped during validation, and their output is multiplied by one minus the percentage of nodes clamped during training.

Convolutional Neural Networks, CNN A common pattern recognition technique in computer visualization. Using a 2D black-and-white image as an example: A kernel of size $m \times n$ is convolved over the input image and an output is produced. This is done by pointwise multiplying the kernel over the input and summing the output. The kernel is moved over the input with some specified stride, often 1. The kernel output is maximized when all the input values match their corresponding kernel value. In essence, the output is a map of how well the kernel fit the input image at different coordinates. For instance, the kernel $[-1/2, 0, 1]$ would give a high output wherever the input vector changes from high to low. This particular kernel is used to detect vertical edges in an image.

It could be argued that several fully connected layers or sparsely connected layers can do the same. However, convolution has the advantage of having far fewer weights to tune.

CNNs are also translation invariant, due to the use of pooling. Pooling techniques are essentially convolution, but the weights are replaced with a non-linear transformation, such as max or min.

Max-pooling is the most common pooling variant in this study. It takes the maximum value from the convolution map within a specific window which is moved across the output with some specified stride and dimensions.

The equation for a 1D convolution is shown in 2.7. Y is the downstream layer, the convolution output. The input from the upstream layer is denoted as X. The kernel, w, is the weights that are multiplied with the input. The kernel width is d, and σ is the point-wise multiplication iterable.

$$Y(k) = (X * w)(k) = \sum_{\sigma=-d}^{d} X(k+\sigma)w(\sigma) \tag{2.7}$$

The idea is that a convolution increases the abstraction of the input. A CNN is capable of *hierarchical feature detection*, where each convolution creates a new feature map. Chaining convolutions reduces the input fidelity while identifying increasingly higher order features.

While known mostly for its success in computer visualization, CNNs is performant on NLP tasks as well [24]. In such tasks, the kernel tensor is of size $m \times d$ and is commonly referred to as a *filter*, since the input is one dimensional. m is the filter width, how many words the filter can see at a time, and d is the embedding dimensionality. An example of the CNN architecture is shown in figure 2.1.

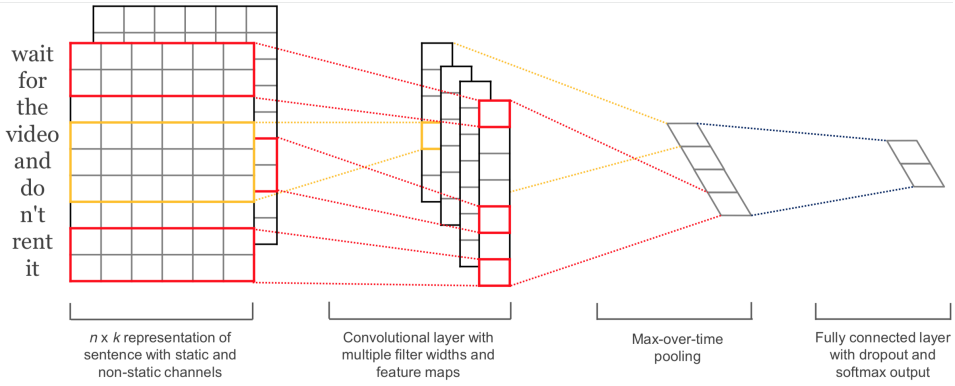


Figure 2.1: An example model overview of a CNN with with two channels for a sentence [1], with four filters and two target classes.

Embeddings Words must be encoded before they are passed to a neural network. One-hot encoding is sometimes used for this; a vector v is initialized. Each element of this vector is a boolean value, representing the presence of a single word from the input. A major constraint of this representation is the *curse of dimensionality*. $|v|$ is equal to the set of words for the encoder, i.e., the dimensionality is $|v|$. Because of this and the fact that the values are boolean, the distance between two different instances of this vector will always be equal to $\sqrt{2}$. Embeddings is a technique to alleviate this effect. An alternative representation for the input is created by using hidden layer activation patterns. This reduces the dimensionality from $|v|$ to something computationally tractable. Word2vec [25], fasttext [26], and GloVe [27] are some of the earlier off-the-shelf available word embeddings, reducing the dimensionality to a range between 50 and 300. These did not embed context for the words, e.g. *Apple* in *Apple computers* would be embedded the same way as *fresh apple*. ELMo [28] and BERT [29] are able to capture this context sensitivity and are now considered SOTA. These embeddings also have out-of-vocabulary mitigation. They also have generally higher accuracies on a range of NLP problems. E.g., swapping word2vec with BERT is considered a low-hanging-fruit to improve model accuracy. The dimensionality of these are higher, around 1000, but still computationally tractable. This increase in dimensionality also means that the interpretation or explanations of the embeddings will be worse, and might completely offset the increase in accuracy with a significant loss of explainability.

When training the embeddings, two important properties are approximated. The distance between similar words in the vector space should be close. This means that the distance is not always $\sqrt{2}$. The word embeddings should also have semantically meaningful vector operations. The textbook example of this is *King - Man + Woman = Queen*[1].

It is important to note that an embedding is a single point in a space, while a word is a single unit. A dictionary contains only valid words, while an embedding can be a point in the embedded space with no natural meaning. I.e., a word has an exact embedding,

[1] A recent paper [30] stirred controversy over word embeddings, discussing similar examples that are biased and that embeddings, in general, are biased through our language.

but an embedding does not necessarily have a corresponding word. Another detail is that the distance measures replaceability, not similarity. For instance, *flavored* and *unflavored* would be very close, even if they are considered opposites by humans.

Embeddings are not restricted to the domain of words. Embeddings can be used to map arbitrary integers to some latent space. This is useful, for instance, when encoding categorical values to a vector space which is compatible with the rest of the neural network.

Out-of-vocabulary, OOV Embeddings in NLP maps words or characters to some feature space. When doing so, only a subset of the input language is considered. Rare words and spelling mistakes can, therefore, be unknown to a specific embedding. Such words will typically be ignored by the embedding and skipped in its entirety. A common approach to this issue is to use character level embedding instead. Newer word embeddings can dynamically capture the context of unknown words by using the contextual information of the other words in the sentence.

Attention Attention is a technique which can be used to aggregate dimensions of data as an alternative to max-pool, average-pool, or a similar operation. It is based on the idea that humans do not consider every piece of the input equally. Instead, we rather choose where to direct our attention depending on the task at hand. It can be implemented by having a, preferably small, neural net in the model to compute weights for a weighted average of the input. It has been used successfully for several tasks involving sequence modeling [31–33].

Neural Collaborative Filtering, NCF Inspired by the ability of deep learning to model complex relations in data, He et al. [34] describes Neural Collaborative Filtering as a generalization of latent factor models 2.1.2. When using multiple layers in the model with non-linear activation functions, more complicated relations in the user-item matrix can be modeled. Figure 2.2 shows a framework for NCF.

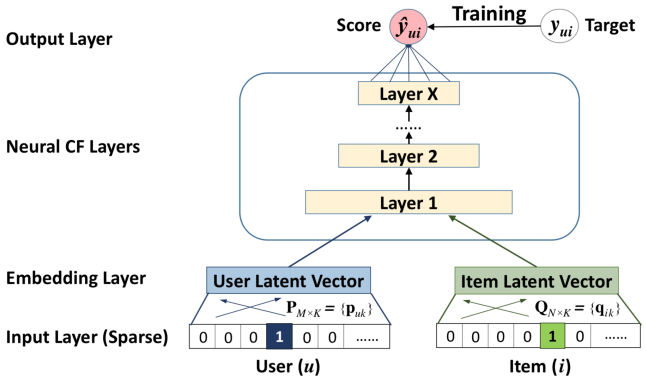


Figure 2.2: Neural Collaborative Filtering framework. Figure from He et al. [34].

2.3 Machine learning

2.3.1 Metrics

Evaluation is a key component in supervised algorithms, both for training a single model, and for comparing different models. Hypotheses about components of the model can also be tested by assessing their contributions to the model. Such metrics can be divided into two categories: offline or online. Offline metrics are performed on a dataset, while online metrics are extrapolated from the behavior of the users using the system.

Evaluating the performance of recommendation systems is commonly done through offline metrics such as RMSE,

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{T}} \tag{2.8}$$

and MAE:

$$MAE = \frac{\sum_{i=1}^{n}|y_i - x_i|}{n} = \frac{\sum_{i=1}^{n}|e_i|}{n}. \tag{2.9}$$

These metrics measure the error, or equivalently, the distance between the predicted value from the model and the real value.

It is common to split the data into a test set and a training set. The system is trained on the training set before it is tested on the unseen test set. The test phase aims to estimate the real world performance of the model. This process is a general strategy in machine learning that is done to avoid overfitting. Overfitting is discussed later in this section.

One way to split the data is to choose some users and select N ratings from them as the test set. Another method is to remove some portion of all ratings. A ratio is chosen as the amount of training and test set, e.g., nine parts training and one part test. *K-fold cross validation* is an extension of this, where the data is split into k parts. Model training is run on $k - 1$ folds, while the last fold remains excluded as the test set. The process is repeated k times until each fold has been the test set exactly once. This is done to increase the accuracy of the test error.

Early stopping in deep learning also uses the validation set. The validation score is measured between each epoch (a complete pass over the training set) which often has a U-shaped figure. The training stops early as soon as the validation score starts to increase.

The evaluation can also be seen as a classification problem instead of a regression problem. This is done by choosing the top-N predicted recommendations and label them as the positive class. These instances are compared against the actual top-n for that user, and we can calculate other metrics [35] such as precision, recall, F-measure, MAP, and MRR.

As mentioned, testing the model accuracy on a separate data set is done to identify overfitting. This phenomenon occurs when a model is tuned too close to the training data, resulting in poor generalization on new, unseen data. Such models have low bias on the training data, but high variability on new data. Underfitting is when the model is too simplistic and does not capture the general trend. The textbook example of this is a linear regression of a polynomial. In this case, the model has low variance but high bias: the

(a) Alt 1 **(b)** Alt 2

Figure 2.3: Two possible configurations of an AB test for recommender systems

opposite of overfitting. An ideal model would have both low bias and low variance, but a balance must be struck between the two. This is called the *bias-variance tradeoff*.

There exists no widely used offline metric for explanations yet. Informally, it is our impression that XAI articles often use sensitivity analysis on the input data, which is tied to the explanations they generate. Others measure the performance of the model through the loss of information when removing features from the input, often by measuring the loss of accuracy.

A/B Testing This is a measure of the impact of using system A instead of a different system B. In general, this is done by presenting either system A or system B to different users in combination with online metrics to measure the performance of each system. The accuracy of the metrics increases as more and more users are tested.

Recommender systems can have a different approach. Typically, more than one item is presented. The system can use both model A and model B to recommend items to the user.

Two ways of presenting this to users in a shelf layout are illustrated in figure 2.3. The letters represent the items recommended by that model. As an example, Netflix uses the second approach [8].

Some standard online metrics used as measures in A/B testing include Click-through rate, equation 2.10, and View-through rate, equation 2.11. Such metrics allow us to evaluate the performance of models without the need for explicit feedback. User-facing explanations are often tested with A/B testing since offline metrics are unable to capture different aspects of HCI such as trust.

$$\text{CTR} = \frac{\text{Number of click-throughs}}{\text{Number of impressions}} \times 100(\%) \tag{2.10}$$

$$\text{VTR} = \frac{\text{Number of view-throughs}}{\text{Number of impressions}} \times 100(\%) \tag{2.11}$$

2.3.2 Curse of dimensionality

High dimensional data introduces a handful of problems in machine learning. If the number of data points is lower than the number of dimensions, overfitting will be an issue, as it is possible to cherry pick the features that match the target variable in the dataset. Another problem is that distance metrics such as Manhattan or Euclidean distance gets increasingly worse as the dimensionality increases. The data points become uniformly distant from each other, which makes proximity-based clustering difficult. As mentioned in the previous section, one-hot-encoding suffers from this phenomenon. *Dimension reduction*

techniques attempt to alleviate this effect. *Principal component analysis*, PCA, transforms the feature space into an independent orthogonal basis set, which has a lower dimensionality. *Autoencoders* in deep learning can also reduce the dimensionality, but these can have nonlinear feature maps, in contrast to PCA, which has linear feature maps.

2.3.3 Clustering

Insight can be gained by grouping unlabeled data points together. In this unsupervised approach, humans would manually annotate the resulting clusters, e.g., identifying customer segments. A pairwise distance metric is used to rank the similarity, or dissimilarity, of data points. Euclidean or Manhattan distance is often used for this. While intuitive for low dimensions, these distances become inflated if the number of dimensions is high, as mentioned earlier. The ranking method and its general algorithm are also of interest, as these make different prior assumptions about the data set. One important factor is the assumption related to the shape or geometry of the clusters. K-NN typically favors globular clusters, while hierarchical clustering can be used with non-globular data. Another factor is the tuning parameters, often directly related to how many clusters the data set has. This is often something that must be known prior to the clustering, but can be estimated by a measure of the cluster fitness, such as the Silhouette score [36].

2.4 Explainable AI

Explainable AI methods are divided into two categories. Model-based and post-hoc. The former are methods that devise interpretable models; the predictions are explainable by looking at the model. E.g., a linear model is interpretable by inspecting its intersection and slope in the units of the axes. Similarly, a reasonably sized decision tree is also interpretable. For recommender systems, content-based systems are usually model-based. E.g., a user and item profile-based system with a distance metric as a similarity measure can produce an explanation for a recommendation by comparing the user and item profile.

Many models are not interpretable, and explanations must be approximated to this *black box*. These are defined as post-hoc explanation-based systems. Latent factor models, for instance, do not have intuitive features which makes interpretations difficult. However, linear models can model some portion of the information captured. Decision trees can also be used to find post-hoc explanations for model choices.

In addition to dividing the explanation methods into model-based and post-hoc, it is also useful to consider the difference between local and global explanations. Local explanations are those which explain a single data point, e.g., a specific recommendation for a user. Local explanations can be true for a specific data point and its neighbors. However, these explanations may be inconsistent for other neighbors that are considered similar by users. These explanations can, therefore, lead to a reduction of trust. Global explanations are based on how the model works in a general sense. I.e., the explanations are true for the majority of the data points in a class.

Evaluation of explanations is a new and uncharted research area. It is not common practice to evaluate the explanations produced by explainable recommender systems with offline measures. The impact of the explanations is instead tested through A/B testing

or case studies. To our understanding, offline metrics are often tightly coupled to the explanation algorithm in such a way that the metric is moot from a human interaction standpoint.

LIME *Locally Interpretable Model-agnostic Explanations* [37] trains an interpretable model on the neighborhood around the prediction. The explanation from the system is based on the interpretable model instead of the original model. The interpretable model is chosen by weighing its complexity against its ability to find recommendations similar to the global model in a neighborhood around the prediction. The neighborhood around the prediction is produced by making small changes to the input parameters of the model and observing the new predictions by the model.

In a setting with explicit feedback, the rating of an item by a user is altered, and the predicted rating is observed. A problem when applying this approach to recommender system is that the recommendations are based on a possibly complex relation between the ratings, so if any rating is changed, it could produce an ”unrealistic” user to which the global model gives poor predictions. In this case, the interpretable model could be similar to the global one, but the explanations it provides may be of poor quality. More accurate explanations could be obtained by changing multiple ratings simultaneously, but this quickly becomes computationally intractable. It could be argued that ratings can be grouped by clustering them in a preprocessing step, e.g., by using topic representations from Latent Factor models(see chapter 2.1.2). This could improve the performance of LIME in our case.

In the domain of reviews as user feedback, the reviews can be perturbed either by perturbing each review of the user, or by perturbing the set of reviews for a user or an item.

2.4.1 Feature importance plots

One approach to post-hoc explanations is by using feature importance plots. The general idea behind feature importance plots is to visualize the size of the impact that each feature has on the target variable. Two common examples of feature importance plots are *partial dependence plots* and *permutation importance plots*. Both of the methods are based on a matrix of labeled data points and a trained model. The explanations assume that the data points are along the rows and the features along the columns of the data matrix.

Partial dependence plots The idea behind partial dependence plots [38] is simple. The range of values for each feature is determined, and a single data point is chosen at random. Each of the variables then takes different values within their range while the model prediction is calculated. The remaining variables remain fixed. This creates a single plot for the effect of that variable on the response variable. By repeating this for all variables, we can plot a local explanation for this data point. The process can also be repeated for all rows, producing a global explanation.

Permutation importance plots Permutation importance [39] is performed by first choosing a known target variable. The model is used to predict this target variable, and the accu-

racy of the model is calculated. Recall that our known data is represented as a matrix with data points, users, along with the rows and features along the columns. The values of each feature column are then, in turn, permuted randomly, and the model predicts new target variables for each user. The feature importance score of this feature is the decrease in the accuracy of the model. The ordering of the feature column is restored before evaluating the next feature.

Using either partial dependence or permutation importance plots to explain recommendations from a recommender system is challenging. One of the problems is that the predicted rating of an item is rarely dependent on a few existing ratings, but rather a combination of several items. This means that a predicted rating might not change when a variable of its item is changed. If the prediction does change, the difference is most likely small. This produces a situation where each rating seems to take on a binary value of either essential or not, and the plots will not reveal much insight. A subset of the ratings should be evaluated at a time in order to gain more information from the methods, but this is computationally intractable.

However, in some domains, these methods can be used to provide explanations to the user. Permutation importance plots might be more informative than a partial dependence plot since it only finds one number per feature. It is easier to present numbers to users as these can be aggregated, while the partial dependence plots cannot be aggregated in the same way.

2.4.2 Deep learning

Deep learning-based models are usually considered black boxes and explained using post-hoc models. These post-hoc models can, as with other explanations, be divided into global and local.

Saliency maps A technique to determine and visualize the impact each pixel of an input image has on the class score function is called saliency maps [40]. A forward pass over the network with the considered input image is computed to find the class score unless it is already known. A backward pass over the network back to the input layer is then performed. The partial derivative that is computed for each input pixel in the input layer is then the saliency value for that pixel. The value for each pixel can then be used to create a greyscale image called a saliency map. Since the technique only requires a single backward pass, it is computationally cheap. The saliency maps themselves often resemble the shape of the class in the image, which makes them easily understandable for humans.

The Layer-wise Relevance Propagation *Layerwise Relevance Propagation* [41] is a technique which finds the *relevance* or contributions of input features towards a classification. The idea is that the target probability is traced back through the neural network while preserving the total relevance in each layer. This is shown in equation 2.12 with an illustration in figure 2.4. The output of the last layer, $f(x)$, is computed during the forward pass. $R_k^{(l+1)}$ is defined as the **R**elevance for the **k**-th node in the **l**-th layer. $R_{j\leftarrow k}^{(l,l+1)}$ is defined as the **R**elevance propagating from the **k**-th node in the **l+1**-th layer to the **j**-th

node in the **l**-th layer. The relevance $R_k^{(l+1)}$ is distributed over all its inbound neurons by multiplying the output with each of the nodes' contribution. This is measured as a product of the node weight, w_{jk}, and its activation, a_j, divided by the sum of all contributions from the previous layer: $\sum_h a_h w_{hk}$.

$$R_{j\leftarrow k}^{(l,l+1)} = R_k^{(l+1)} \frac{a_j w_{jk}}{\sum_h a_h w_{hk}} \tag{2.12}$$

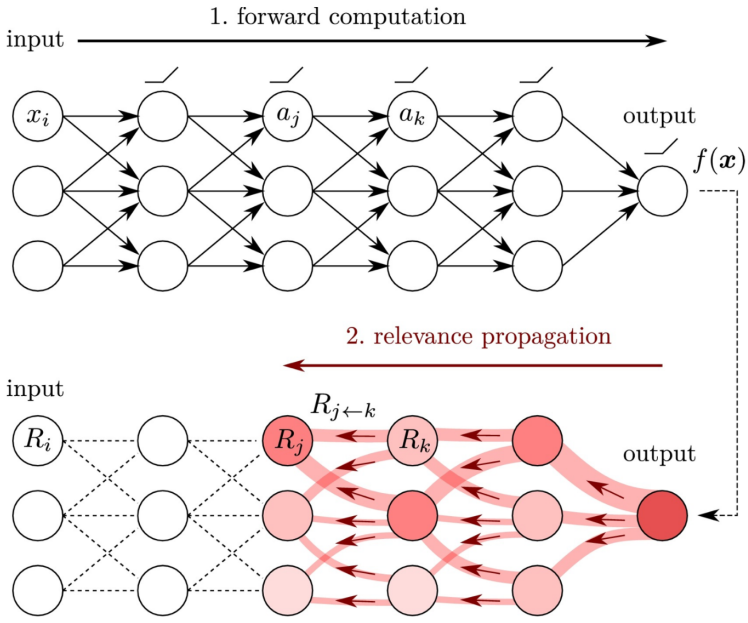


Figure 2.4: The probability, $f(x)$, is distributed over the inbound edges. This is a recursive process which repeats until reaching the input features. [42]

Testing with CAV, TCAV Been Kim et al. [2] introduces *Testing with Concept Activation Vectors*, TCAV, as a method to quantify the impact of user-defined concepts on the predictions from a neural network model. Concepts are defined as a set of model inputs where each model input expresses the concept. These concepts do not have to be specified during the training of the model. Next, one layer of the model under inspection is chosen as the space where the concept activation vectors are computed. How close this layer is to the output layer often affects the accuracy of the method, and should be chosen depending on how abstract the concepts are. The authors of the method evaluate the impact of the chosen layer further. Concept activation vectors are defined as the vector orthogonal to a linear classifier which separates the activations from the concept input data and activations from a random set of input data. The *conceptual sensitivity* of a class to a concept is the directional derivative of the prediction w.r.t the concept activation vector. The score of a concept is then the fraction of examples which had a positive derivative.

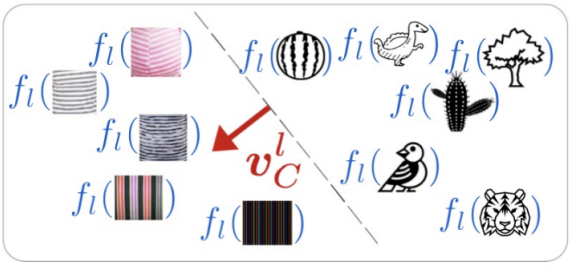


Figure 2.5: The linear classifier and its orthogonal concept activation vector, from [2].

This method is based on user-defined concepts. One of the examples used in the paper is: How important is the concept *stripes* to images classified as zebras? This makes the idea of a concept visualization task somewhat concise, but not well defined in other domains such as NLP. Deciding the kind of concepts to test and which inputs that represents considered inputs is not trivial for recommender engines. If the recommender system is only based on ratings, there may not be any meaningful concepts in the input data to test. However, if the recommendations are based on item reviews as well, concepts found in the reviews can be considered. As an example, these concepts can be topics the reviews mentions. Figure 2.5 illustrates the linear classifier and the corresponding concept activation vector.

2.5 Explainable AI and recommender systems

The combination of XAI and recommender systems is of our interest. Classical content-based recommender systems are interpretable by nature, but advances in deep learning have made complex models with higher performances more popular. Therefore there is a need for explainable recommender systems with high efficacy. Some of these are surveyed in section 3. The theory for the model we decided to inspect, NARRE, is presented below.

2.5.1 NARRE

Motivation Neural Attentional Rating Regression with Review-level Explanations [4], NARRE, is, as the name implies, a review-based recommender model. Ratings and reviews from the users for items is the basis which the model uses to predict user-item ratings. The primary motivation is that current review-based models fail to assess the *usefulness* of individual reviews. The authors show that this is indeed a contributing factor to the performance of the model. This usefulness measures how informative the review is to the rating predictions.

Architecture In this model, a user component and an item component is computed separately and combined to find the final rating. The architecture of these two components are identical. They do not share weights. The network is given several three tuples on the form *(user-ID, item-ID, review text)*. The user ID is always the same in the user component of

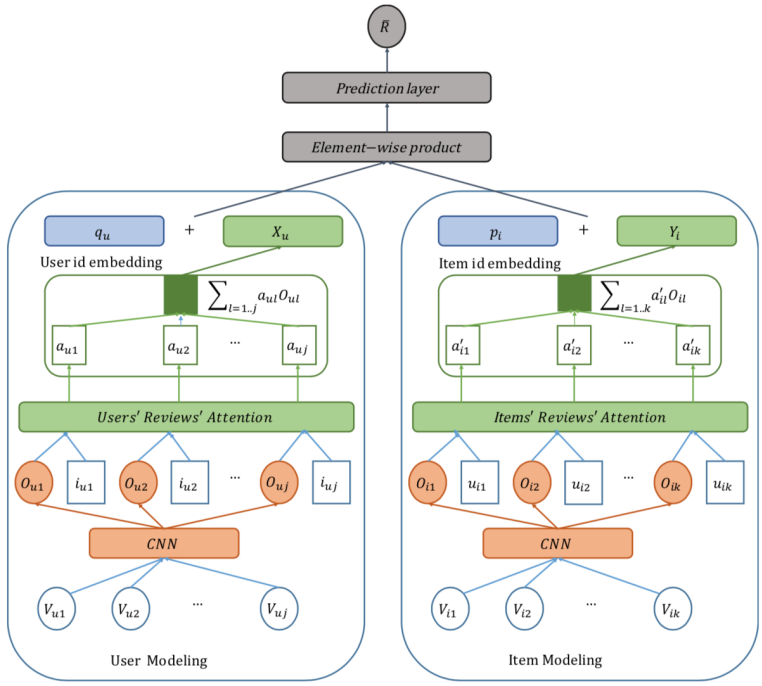


Figure 2.6: A complete overview of the architecture.

the network, while the item ID is always the same in the item component of the network. This dual split creates an item profile and a user profile based on the input. An overview of this is seen in figure 2.6.

A pre-processing step normalizes the review text. All strings are converted to lowercase. Numbers, symbols, and words not present in the embedding are removed. word2vec [43] is then used to embed the words. Stopwords are kept, rather than removed. The embeddings are processed by a CNN consisting of 100 3-width kernels before being max-pooled. An overview of the CNN architecture is seen in figure 2.7. An attention layer weighs the output of the CNN and an embedding of the IDs so the network can ignore reviews from specific users and emphasize reviews which better characterizes items. The final component is based on neural collaborative filtering, which is described in section 2.2.

Explanations NARRE uses the reviews of the item to explain the prediction. As the model weighs each review as part of the inference, it can use the attention weight for each review as a measure of usefulness and present the review with the highest attention to the user as an explanation for the recommendation. The authors point out that text-based explanations can be very powerful, and brings up that machine-generated text is not as convincing as human-written text. This motivates the use of the user-written reviews from the model input as explanations.

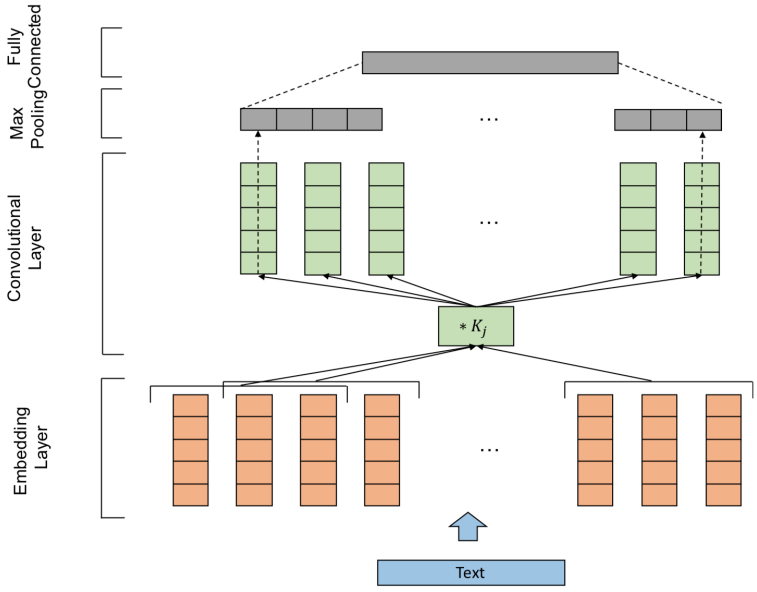


Figure 2.7: Overview of the CNN component of the architecture

Chapter 3

Survey

There are several approaches to explaining the recommendations made by a content-based recommender in the literature. In section 3.1, we will present some of these. In section 3.2 we present some of the datasets considered for implementation purposes.

3.1 Related work

The majority of the contents of paragraph **ProPPR**, **Ripple Net**, **Explainable matrix factorization**, **TriRank** and **AMF** has been copied from our pilot project [44] that led up to this project.

ProPPR Catherine et al. [45] models items and other entities for the user in a knowledge graph. Each node in the graph is based on its child nodes, similar to an ontology. Each movie can then be expressed as a sum of nodes where each node contains an explanation. The method is based on ProPPR [46], programming with personalized page rank. The likes and dislikes of a user are fed into the system as weight initialization. These weights are then propagated through the knowledge graph. The recommendation is then only based on the preferences of the user, which makes it possible for the user to inspect the data the system bases the recommendations on, and make adjustments according to the current mood. All paths connecting the initial preferences with the recommended movie can be listed as an explanation for the current movie recommendation.

However, there are some problems with this approach in our domain. Most users have either too little explicit feedback for high-quality recommendations or too much explicit feedback for intuitive explanations which are easy to act upon. For the users who have given a large amount of explicit feedback, the influence ratings have on explanations become less intuitive, which makes it harder for the users to update their likes and dislikes in the system to reflect their current mood. Another issue is that it does not leverage implicit feedback from the user. Ignoring this worsens the quality of the recommendations, especially for users with a low number of explicit feedback.

Ripple Net A different solution is proposed by Wang et al. [47]. They present a framework called RippleNet. This framework is based on a knowledge graph, where each node is an entity and the edges are relations between the entities. For example, two nodes could be an actor and a movie, with an edge between them if the actor played in the movie. Implicit feedback from the user is used as a seed in this graph as the starting nodes to an n-neighborhood algorithm for recommending an item. The information from the implicit feedback *ripples* through the graph, which can be conceptualized as a preference propagation.

Explainable matrix factorization Collaborative filtering based on matrix factorization is common in recommender systems. Adding ad-hoc explanations to such systems can be a fairly straight-forward task. Abdollahi and Nasraoui [48] propose a CF technique which computes explainable recommendations. They add a regularizing term based on the ratio of the nearest neighbors that have rated the item to the total amount of nearest neighbors.

This technique aims to improve the explainability of the returned items. This shows that explainability can be built into a model by introducing an explainability metric as a part of the loss function. It has to be combined with a different technique to form a complete recommender system with explanations since it does not produce any explanations itself.

TriRank He et al. [49] devises an algorithm, TriRank, which produces recommendations and explanations based on textual reviews and ratings. It relies on a preprocessing step, a state-of-the-art aspect extraction tool, that reduces review text to aspects. The product of this is feature-opinion pairs with an associated binary sentiment, i.e., triplets on the form `(feature, opinion, sentiment)`. E.g., `(service, excellent, positive)`. The features are nouns and are interpreted as aspects. An undirected weighted tripartite graph is constructed initially. The vertices are the union of the set of users, the set of items to be recommended, and the set of aspects which were extracted in the earlier step. All the edges are between vertices of different sets since the graph is tripartite. Interactions are stored as edges from user nodes to item nodes. The weights of these edges are ratings. The ratings are scaled as part of the algorithm so that they can be either implicit or explicit. An aspect used by a user in any review is stored as an edge from that user node to an aspect node. Similarly, the edges between aspects and items represent aspects that have been used to describe each item. The edge weight between aspects and users or items are the number of occurrences in reviews from users or about items, respectively. These weights are normalized.

When a recommendation is requested, the prior preferences of the user is set as initial weights, or scores, to the user, item, and aspect vertices. The weights of the item vertices are set to the rating of that user to that item, the same as the edge weights from the corresponding user vertex. The weights of the aspect vertices are set to the same as the edge weights from the user vertex to the aspect vertex. For the other user vertices, social information can be incorporated, and a relation score can be set to each vertex, but the authors do not use such information and set the weight of the user requesting the recommendation to 1 and the weight of all the other users' vertices to 0. Then, the TriRank algorithm updates all the vertex weights in the graph iteratively according to two principles. The

principle of smoothness states that the weight of connected vertices should not vary too much. The principle of fitting constraints states that the updated weights should not differ too much from their initial values. A loss function which is minimized expresses these principles.

Aspects and unseen items with the most significant weights can be presented as explanations and recommendations, respectively after the algorithm has converged. The algorithm could be reduced to collaborative filtering by removing vertices and edges associated with aspects. The aspects add information which both improves the recommendations and functions as explanations. The user can update the weight assigned to an aspect to update the recommendations since the aspects are closely tied to the recommendations. This makes the recommendations scrutable.

AMF Hou et al. [50] also incorporate reviews into their explainable recommendation system by extracting aspects from the reviews. Their definition of aspects is different from how He et al. [49] defined it. Here, aspects are similar to topics as we defined it in our description of latent factor models in section 2.

Aspect Segmentation Algorithm [51] is an algorithm which maps the words in each review onto aspects. Based on these aspects they create *User Aspect Preference* (UAP) and *Item Aspect Quality* (IAQ) matrices. Such matrices define the user preference to an aspect and the review sentiment of an item to an aspect. Predicted ratings are made from their proposed *Aspect-based Matrix Factorization* (AMF) model. In addition to assuming that two latent factor matrices can reconstruct the original rating matrix, it also represents the latent factor matrices as two different latent factor matrices which map user preference on aspects and item sentiment information on aspects to latent space. These four latent factor matrices are trained simultaneously using stochastic gradient descent. The motivation is that the information extracted from aspects should improve the accuracy of rating predictions.

Explanations can then be visualized by plotting the row in the UAP matrix corresponding to a user and compare it to the row in the IAQ matrix corresponding to the recommended item.

The authors tested the algorithm on a hotel review dataset and a video game review dataset, where the algorithm performed well and produced informative explanations. The aspects are entirely dependent on the domains. It is fair to assume that even though those domains are a little different, the algorithm would probably work well on the domain of movies and TV shows.

CNN filter interpretation The authors of [52] created a method to identify important features for a CNN in the setting of NLP. They calculated the correlation between all triplets in the dataset and the target classes. This makes it possible to find a threshold for each filter for how important triplets must be for them to count as contributing to the given classification. The importance of a single triplet was measured as its purity of target classes. Thresholding acts as a regularization technique, refining the input features, the triplets, to make the explanations more human interpretable. This was done because embeddings match multiple words, but to different degrees, resulting in many possible triplet matches.

The remaining triplets after thresholding were aggregated based on the hamming distance for the slots in the filter, with a distance of 0, 1, and 2. Each slot is a position in the filter. This makes it possible to identify filters which has one or two free slots, i.e., slots where the filter is indifferent to the input given the non-free slots. Triplets can then be sorted by contribution based on their contribution score to the target class. Finally, the triplets were clustered using mean shift clustering on the word embeddings of the triplets. Mean shift clustering is not dependent on prior knowledge of the number of clusters, which is useful in this case since the cluster count is unknown.

DARPA DARPA [53] have started an Explainable AI program [54] with two goals. It aims to create a suite of machine learning techniques that produce more explainable and high-performing models, and to enable humans to understand, trust, and manage artificially intelligent partners. DARPA has not yet released any techniques or models from this program. They explore XAI within, amongst other areas and topics [55]; representations, methods of learning, architecture, user interactions and decision making, acceptance testing, question answering systems.

3.2 Data sets

The majority of the contents from this section has been copied from our pilot project [44] that led up to this project. We considered different datasets for different recommendation systems. Some of the properties we evaluated were sparsity, quality, type, and license.

Internet Movie Database, or IMDb[1], provides metadata which can only be used for personal and non-commercial usage but offers commercial licenses if necessary. All metadata is categorized by type and is available as compressed files from their website. To the authors' knowledge, this is by far the most comprehensive metadata set.

The Movie Database, TMDb[2], also provides metadata with similar restrictions. This data is accessible through a REST-API, not as handily compressed files. All metadata must be scraped from their API, within a specified rate limit restriction. The number of API-calls necessary to scrape all metadata were unfeasible at the time. Therefore we did not explore this dataset further.

Opinion mining and sentiment analysis could be used to extract more data about a movie, such as common words or phrases. Sadly, the IMDb dataset did not contain user-written reviews. Furthermore, they do not permit any form of scraping of these reviews from their website. TMDb supplies and permits such usage for its REST-API. We scraped all movie reviews from TMDb for future purposes. It should be noted that an immediate issue with this dataset was the sparsity. Around 3600 reviews were scraped from all movies available. IMDb has in comparison vastly more data, where a single movie can have 3000 reviews.

Tweets can also hold relevant information in a short and concise form. We got a Twitter developer account approved and explored the API of Twitter [3]. One of the problems that

[1] https://www.imdb.com/

[2] https://www.themoviedb.org/

[3] https://twitter.com/

arose was how to match movies against Tweets. The API must be queried in such a way that one gets as many relevant Tweets as possible. This could be done by querying on movie titles, actors, or keywords in general. We quickly discovered that movie titles are ambiguous, as there is no standard for these titles. Movies can have multiple titles in different languages. Many movies can also share the same title but have a different release year. The operation of linking movies from one dataset to another is problematic and affects the quality of the combined dataset.

The movie title mismatch issue was also encountered when we found an Amazon movie review dataset[4]. Although this data is cleaner, movie title ambiguity would still be an issue affecting precision and recall when joining the datasets. This dataset contained 8×10^6 reviews with ratings for movies up until the year 2012. It also contains explicit feedback on the reviews themselves in the form of *helpful* and *not helpful* count.

A promising dataset is Movielens [56]. It is an ongoing project from the GroupLens Research team [57] and is free to use for non-commercial usage. Movielens provides movies, users, explicit ratings, and links to both IMDb and TMDb, but no metadata. This is in contrast to IMDb and TMDb, which has metadata but no ratings. Links between this dataset and IMDb or TMDb makes it possible to implement some of the algorithms mentioned earlier and allows the possibility of comparing results against other works. Grouplens also provides [58] movie tags in the Tag Genome Data Set. Movielens users have provided tags for many movies, and the project authors have calculated the relevance of each tag for all movies. See table 3.1 and 3.2 for statistics on the Movielens and Tag Genome datasets, respectively. Movielens is one of the few datasets which provides explicit ratings, but sparsity remains an issue.

Netflix[5] held a competition called *Netflix Prize* [59] for recommender systems in 2008-2009. In short, the competition was to beat Netflix's algorithm for predictions. They provided a dataset of 1×10^9 ratings from around 5×10^5 users on 17×10^4 movies. This makes this dataset far less sparse than the Grouplens dataset. The number of movies was also comparatively lower than Grouplens. The license was restricted to non-commercial usage, and it is improbable to get a commercial license for it. Therefore, we decided not to use it for further work.

The Amazon product data dataset [60–62] on the Movies and TV category in its *5-core* version were used in this project. *5-core* means that the data has been reduced to only include the users and items which has more than five reviews each. The dataset includes information such as helpfulness votes of reviews, product metadata, and links. In this project, we used the *5-core* and only used user id, item id, rating, and review text. See table 3.3 for some statistics of this dataset. A sample of this data set can be found in figure 6.1 in appendix A. During training the number of ratings and reviews per user were set to be maximally 17 and per item maximally 59. Each review was also limited to only contain 409 words. These values were chosen so that 90% of all users, items, and reviews would not have to be truncated. The result of this truncation and the *5-core* variation is that each user has between 5 and 17 reviews. Each item has between 5 and 59 reviews, and each review has a maximum length of 409 words.

We also explored datasets from other domains. A collection of datasets for recom-

[4]https://snap.stanford.edu/data/web-Movies.html

[5]https://www.netflix.com/browse

Table 3.1: Movielens dataset

Stat	Value
Movies	26×10^3
Users	140×10^3
Ratings	20×10^6
Sparsity	0.9946

Table 3.2: Genome dataset

Stat	Value
Distinct tags	1×10^3
Labeled tags	12×10^6

Stat	Value
Movies	5×10^4
Users	1.2×10^5
Ratings and reviews	1.6×10^6

Table 3.3: Amazon movies and TV 5-core

mender systems has been made by the University of California, San Diego [62]. 225×10^6 user-book interactions are published from Goodreads [63], a book curation service. 82×10^6 explicit ratings and reviews are given in a larger Amazon dataset [60–62] of items such as movies, electronics, and clothes. LibraryThing [64] has a dataset of 10^6 book ratings based on a social context.

Chapter 4

Approach

This section elaborates the approaches chosen in this work.

There are several deep learning-based models for recommender systems. Since we did not want to spend time reinventing the wheel and implement an existing model, one criterion when choosing the model was the availability of detailed information about the model in the form of either a model description or source code. Furthermore, it must be trained on a publicly available data set. This makes it possible to analyze the input data for interpretations of both the model and the predictions it makes. The model must be SOTA or near so. This lead to the choice of NARRE. This model achieves SOTA performance on recommendations, and the authors provide source code as well. This model depends on ratings and reviews from users. Hence we used the Amazon 5-core dataset (section 3.2).

As discussed in section 2.5.1, the review explanations from NARRE do not provide detailed and specific information as to *why* the item was recommended for a particular user. We believe that it is difficult to understand the decision process of a system from a single review. It provides too much information while at the same time being too subjective since the review, in many cases, only represent one point of view. This is counter-intuitive of how explanations should be; simple and direct.

We, therefore, examine NARRE with two different approaches. An overview of this examination scheme is described in figure 4.1.

TCAV is used in different ways to identify global concepts that affect the recommendations. The term *concepts* from TCAV in recommender systems is vague and we explore different definitions by applying them to NARRE. NARRE is also decomposed into separate parts, rather than looking at the model as a whole. We restrict our research to the first layers of the model.

Our motivation is also to inspect *what* the model has learned. Instead of looking at explainability for a single prediction, we can look at the general knowledge that lies in the model. This can, in turn, be used to explain predictions.

Section 4.1.1 looks for explanations of the model by using TCAV. This method is used with different combinations of input data to inspect the effects on recommendations. Section 4.1.2 looks at the different parts of the model and inspects these individually. A

short description of implementation is presented in 4.2.

4.1 Overview of the study

An overview of the study is shown in figure 4.1. The input data is passed to the model, and a prediction is generated. A technique which uses LRP is dependent on the input data and the output from a specific layer shown as (b). Another technique which uses TCAV is dependent on the input data, the prediction, and the output from a given layer which is shown as (c) in the figure. Finally, we inspect the weights of a layer, shown as (a).

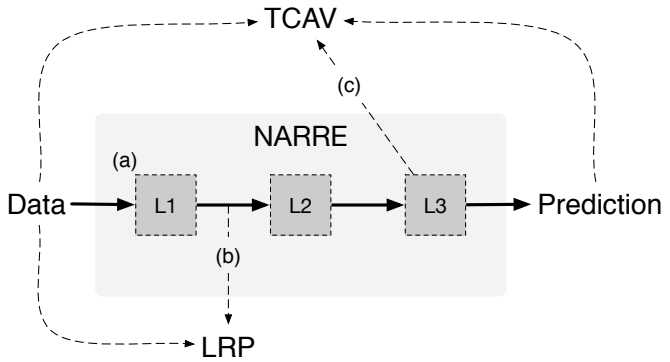


Figure 4.1: Overview of the architecture. (a) represents the analysis of the weights of layer 1. (b) is the output from layer 1 which is fed to the LRP analysis. (c) is the internal representation in layer 3 which TCAV uses for further analysis. Note that this diagram represents an arbitrary simpler model with three layers, w.l.o.g.

4.1.1 TCAV

Since TCAV only requires a set of example inputs to define a concept, we could be quite creative when experimenting with which variations of concepts we wanted to test. One limitation is that TCAV considers the activations in a layer during a regular forward pass of the model. I.e., it requires the same input tensor format. The model relies on user ID and item ID in addition to review text as input. Therefore we used reviews from the dataset when defining concepts. The method would likely work somewhat without the IDs, but we restricted the input to that in the dataset so we would not have to risk performance loss. This problem could also be partially circumvented by considering the activations in a layer which do not base its calculations on either user ID or item ID, meaning that the IDs will intuitively have less impact on the TCAV score. However, this is not a perfect solution since the TCAV score is also dependent on the prediction, which naturally is dependent on the entirety of the input.

TCAV computes the scores against a random set of concepts. In all our experiments, we used a random subset of the data for each random concept. However, we could have used the *inverse*, or *opposite*, of a concept. We did not pursue this idea because TCAV

should still work well when comparing with random concepts and also because the *inverse* of a concept can be ambiguous.

Topics as concepts One approach was to define one word as a concept. The motivation was that the concepts could then be interpreted as topics. In order to collect inputs for the different topics, we used TF-IDF to find the reviews which referred to these words often. TF-IDF [65] is a measure of how often a term appears in a review divided by a measure of how often the term appears in the whole dataset. Because a review could span several topics, we also experimented with splitting the reviews into sentences. In order to ensure that the ”topic sentences” uses similar semantics when discussing the topic, we could use a paraphrase detection model (see [66]) to further rank the reviews on similarity. This was, however, deferred to future work because of time constraints.

Movies as concepts A different approach is to consider each movie as a concept. The concept inputs would then be made up of the reviews written about that movie. The motivation is that users writing reviews about a particular movie would use a certain language and that the activations from this language would be an internal representation of the movie. This approach allowed a form of validation, which is further described in section 4.1.1. Based on the idea of movies as concepts, we could also use the average TCAV score for each movie when rating a movie for several users as a similarity measure between movies, and use this information in a clustering algorithm to find similar movies based on their reviews. Based on this, TCAV could be used to extend the recommendation engine to also suggest movies for the user instead of just the rating regression as it does in its current form. This was deferred to future work.

Sentiments as concepts We explored current uses of the *Large Movie Review Dataset* [67]. This dataset contained 25000 reviews with a sentiment rating from 1 to 10, with 1 representing the most negative sentiment and 10 representing the most positive sentiment. There are several [68, 69] freely available implementations of sentiment analysis based on this dataset. We could leverage such models by using the classifications to create sets of input to TCAV. This would produce instances of classes *positive* and *negative*.

Positive/negative words as concepts Similar to the tests run with the topic words, we used a list of positive and negative words from the book Film Studies [70]. We performed the TF-IDF scoring described earlier in this section on the words. The list of positive words are in table 4.1 and the negative words are in 4.2.

TCAV validation We test TCAV by considering movies that the user had seen as concepts. The set of inputs for each concept is then the reviews written by other users for that movie. In order to evaluate the accuracy of the TCAV score given to a movie, we measure the impact of reviews about the movie on the predictions made by the model on several other movies for the user under evaluation. Since the TCAV scores are based on how the model works when predicting ratings, removing the movie reviews from movies with a high TCAV score should affect the average predicted rating over many movies for one

insightful	clever	charming
comical	charismatic	enjoyable
uproarious	original	tender
hilarious	absorbing	sensitive
riveting	intriguing	powerful
fascinating	pleasant	surprising
dazzling	imaginative	legendary
unpretentious		

Table 4.1: Positive words

violent	moronic	flawed
juvenile	boring	distasteful
ordinary	disgusting	senseless
static	brutal	confused
disappointing	bloody	silly
tired	predictable	stupid
uninteresting	weak	trite
uneven	outdated	dreadful
bland		

Table 4.2: Negative words

user by more than the movie reviews from movies with a low TCAV score. This means that we do not attempt to validate the TCAV scores themselves, but rather whether the ordering of the TCAV scores correlates with the order of the magnitude of the changes in the ratings predicted by the model.

This experiment was performed by using the model to predict the ratings of many movies for a user. Then, we compute the TCAV scores for the movies that this user has written reviews about. After this, we remove each movie review in turn from this user and recalculate the predictions of the same movies for the truncated user. The sum of differences per movie review is used as a measure of how vital that particular movie reviews were to the predictions of the model. This was done for several users, and the results were aggregated. We hypothesize that the ”importance” found for each movie would correlate with the TCAV score for that movie.

We explored whether the data could be more correlated if we removed the least correlated users since this is real-world data, and people are known to be quite inconsistent when rating [5]. The percentage of the number of remaining users compared to the total amount of users can then be described as the percentage fit of the hypothesis.

4.1.2 Component study

The intuition for decomposing the model into smaller components is that explanations for upstream layers provide the basis for downstream layers. Any explanations targeted at the NCF (section 2.2) component of NARRE will be dependent of the input to this

component, which is the sum of user/item-embedding and a hidden layer. Both of which are not trivially interpretable, and must be interpreted as well. The first steps of interpreting NARRE is, therefore, to interpret the model gradually from head to tail. An attempt to explain the NCF-component might result in a neighborhood-based explanation and is left for future work.

The model uses embeddings, CNNs, NCF, and attention layers, but it is not particularly deep. This makes it simpler to inspect the workings of the individual components. Item and user embeddings are used to encode item and user IDs into the network. Therefore they do not provide much information. While the word embedding word2vec is fascinating, it is also challenging to decode because of the latent space. The CNN could provide information. Both by interpreting the weights for the filters, but also by investigating the computation of different words from the input to the output values of the filters. Input such as words is readily interpretable by humans, unlike vectors of seemingly random numbers.

The attention component is the part of the model which currently decides which reviews, or explanations, that are provided to users. The authors of NARRE argue that since the attention layer is trained to determine the usefulness of reviews, the review with the highest attention score should act as a good explanation of the recommendation. They also include experiments to prove their point. In our preliminary studies, we found that a concerning number of recommendations had the highest attention score assigned to an empty review used to pad the user profile. This could be a symptom that our model was underfitted, but it could also indicate that the model eventually learns to rely less on the reviews and more on the ratings when predicting ratings. We did not attempt to define model fitness based on the ratio of empty reviews supplied as explanations since we considered this ratio to be an unreliable and unstable metric.

NCF is the final component of the model. It combines the representations of the reviews from the CNN component weighed by the attention component in one hidden layer and the ratings in another hidden layer. Then, it combines these layers to predict a rating. The authors note that more hidden layers could be added to include additional non-linear transformations. Explanations based on this component could use the weight matrices as a basis for a distance measure and present an explanation similar to the one described in section 2.1.2. There are approaches in the literature to increase the explainability of neighborhood-based system [48], but we did not explore this due to time constraints.

LRP A segment of the network was chosen to limit our scope; we inspected the network from the input layer up until, but not including, the attention layer. This makes the CNN the focus of this subtask. We can interpret the 100 filters on a global level, i.e., see what the model has learned. Investigating the attention layer was left for further work.

We can use these global interpretations to produce local explanations since NARRE provides a review text along with the recommended item. This recommendation can be explained further by producing a set of input features with scores that originate from the review text.

As discussed in chapter 3.1, one method to interpret CNN filter activations is to calculate the correlation between triplets and the target class, set a threshold for *importance*, permute triplets according to the hamming distance, and finally, cluster the triplets.

We decided to use Layer-wise Relevance Propagation, LRP, to find the candidate

triplets, instead of iterating through all possible triplets in the dataset. There would be no need to permute all possible triplets and find the correlation as LRP would assign a score, similar to that of the correlation, to each triplet in the input text. This turned the filter analysis into a regression problem, with the filter activation as the variable being regressed. The idea is that LRP follows the target variable, i.e., the filter activation, backward through the network to the input features. This means that we do not have to find the correlation, as these are similar to our relevance scores from LRP. We could then assign relevance scores on triplets for the different filters.

One of the issues that arose was the lack of a threshold. Each of the 100 filters was activated in 20% of all documents, which means each document had on average 20 filter activations. This was partially solved by binning all triplets by relevance score into each percentile of relevance score. The performance of the model was measured for each of the percentiles by removing all the text that was not present in that percentile. Our motivation for this was the ability to choose a percentile with a reasonable trade-off in model RMSE.

Even after selecting a reasonable threshold by this method, we still had too many distinct words to cluster. We, therefore, selected the top-N words for each slot and attempted to find clusters of these.

This introduced a non-trivial problem: the sorting method. We sorted all triplets by the sum of their slot relevance scores, and gradually removed all outstanding triplets from the corpus and measured the RMSE. The negative relevance score can be negative as well, indicating that the filter can be averse to some words. Therefore we also sorted the triplets on the absolute value of the triplet relevance. We also performed max pooling per filter per document. Multiple triplets in a single document may, after all, activate the same filter. Therefore we choose the highest scoring one to reduce noise. The performance of the different sorting methods are presented in figure 5.8.

In figure 5.10, we inspect the RMSE as a function of the percentile remaining of kept triplets, based on their relevance score. The 100th percentile keeps all triplets, while the 95th percentile removes all triplets that are not in the top 5%.

We also inspected different ways of removing non contributing, or dead, words. The paper we were inspired by used *occlusion*; a method where dead words are replaced with the embedded value of the padding word, e.g., `<PAD/>`. We discovered that some filters were activated by this padding word while inspecting the triplets. Therefore, it could be reasoned that this can impact performance since it generates new triplets. To reduce this effect, we included one more word on each side of a triplet, creating quintuplets instead. These quintuplets were then concatenated together. We also looked at the effect of not including these neighboring words and just concatenating them. This method also has the effect of generating new triplets. These results are presented in figure 5.9.

Our final method of inspecting the CNN was by multiplying the weights of each filter in the CNN with the words in the embedding space. The candidate words were found by sorting the result by the sum of the product. This is similar to how the values would propagate through the network during the forward pass. While this is interesting, it might also not provide any useful information since such words could be coincidental.

4.2 Description of Implementation

4.2.1 TCAV

The computations of TCAV scores are described in section 2.4.2. This was implemented to the best of our ability based on the Github repository [71] from the authors. The implementation in the repository, as well as in the paper, were based on a model for image classification. We had to adjust the form of the predictions in the TCAV framework since NARRE is a regression model.

The authors of NARRE have published an implementation of their model [72], which we based our implementation on. The model was trained on an Nvidia Tesla P100 GPU for one day. Further training did not decrease the score on the validation set.

Topics as concepts We used the TF-IDF implementation in scikit-learn [73] to score the reviews on TF-IDF. For each topic word, we ranked every review based on the TF-IDF score reported and wrote the 200 reviews with the highest score into a folder. We used a review-to-id dictionary to find the correct user and item ID for the model. We did not use individual sentences because we wanted to avoid text that was separated by a period but would not be considered a sentence. Checking the validity of every review sentence would be time-consuming.

Movies as concepts For this approach, we wanted each concept folder to represent a movie. Each movie had a folder containing its reviews. The input tensor we use for concept reviews in TCAV expects reviews authored by one user on several different movies. However, in this case, it would receive a set of reviews about one movie authored by several users. Since every review is from a user to a movie, the CNN should be prepared to handle the slight variation of the inputs. In addition to this input, the IDs for the user who had written the review and the movie it was written for had to be flipped.

Sentiments as concepts We were inspired by existing [69] models and trained a Keras[1] model on a relevant dataset. This model used GloVe [74] embeddings, followed by a dropout layer with $\rho = 10\%$, an LSTM [75] of hidden size 64, a 1D global average pooling, one dense layer of hidden size 32 with ReLU activation and finally a dense layer of one node and ReLU as activation. We then choose the 200 most positive and 200 most negative reviews. We used a linear classifier as the explainable model component to identify the most contributing sentences, using LIME as discussed in 2.4.2. The idea was that individual sentences have varying degrees of contributions to the class, and we could, therefore, find the sentences with the most positive and negative sentiments. These sentences were used as concept examples for TCAV.

Positive/negative words as concepts We used each word in the list of positive and negative words as topic words. Then we did the same TF-IDF scoring as described earlier in this section to make one folder per word. Then we computed the TCAV score for every topic folder for 1600 users and saved the TCAV result per word per user. After this,

[1] https://keras.io/

we used matplotlib's function to produce box-plots of the scores for the positive and the negative words.

TCAV validation The work required to implement the validation builds on the approach where we considered movies as concepts. In addition to the TCAV scores, this approach required a metric which captures the change in rating if a movie review had been removed from the user profile. This was not difficult to implement using the trained model. We used the implementation of Linear Regression from scikit-learn for the statistical analysis of the results. We removed user profiles to increase the accuracy of the trained linear regression model, as explained in section 4.1.1. The algorithm for removing user profiles was greedy, and would not necessarily find the optimal collection of users in terms of R^2. Devising a better algorithm was not attempted because we wanted the analysis to be more predictable and interpretable, and because it would be non-trivial to design an algorithm which would perform significantly faster than the brute-force approach of searching the entire collection space. Searching the entire collection space would be a subset problem with a running time of $O^*(2^n)$, where n is the number of users. Searching the entire collection space is intractable.

4.2.2 Component study

LRP The iNNvestigate [76] library provided implementations of LRP, but relied on Keras. We, therefore, trained NARRE, which uses Tensorflow[2], and created a surrogate model of the CNN in Keras, which matched the original model as close as possible. Finally, the weights from NARRE were imported into our ablated Keras model. We confirmed that the output matched that of NARRE, before using iNNvestigate on the dataset for further processing. Scikit-learn provided a set of clustering algorithms which we used on the word embeddings of the triplets.

[2]https://www.tensorflow.org/

Chapter 5

Evaluation

5.1 Results

5.1.1 TCAV

Topics as concepts

Figure 5.1 shows a box plot of the TCAV score of some topic words. The topics are sorted by the median of their values. See figure 6.2 in appendix B for density plots of the TCAV scores.

Movies as concepts

Using movies as concepts were explored as part of **TCAV validation**, described later in this section.

Sentiments as concepts

The TCAV scores for the positive reviews found from the sentiment analysis are included in figure 5.1 with the other topic words, to provide context to the values.

Positive/negative words as concepts

Figure 5.2 shows box plots of the distributions of the positive and negative words. The orange line is the median of the data, the box contains the 25th to the 75th percentile, the whiskers extend 1.5 times the width of the box past the box, and the circles are outliers. The rightmost half is the negative words sorted by the median, and the leftmost half is the positive words also sorted by the median. See figure 6.3 in appendix B for density plots of the TCAV scores.

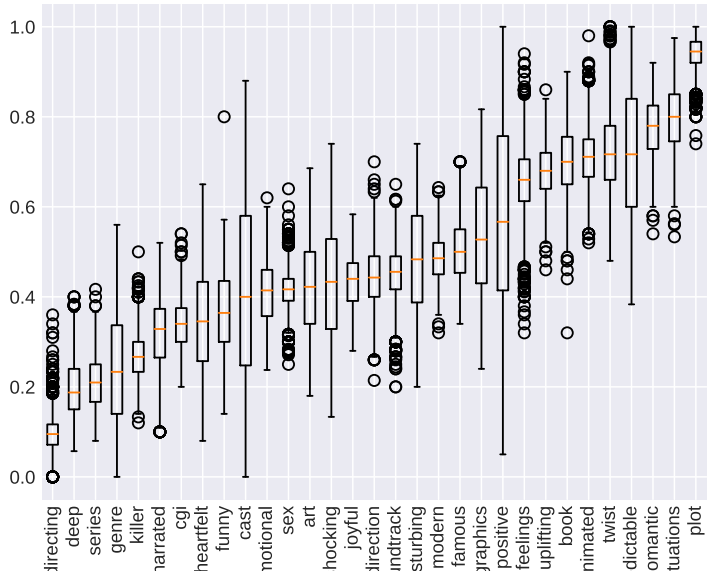


Figure 5.1: Box plot of TCAV score of various topics for several users

TCAV validation

As described in section 4.1.1 we would calculate the TCAV scores for a user per movie and compare that to the change in rating if the movie were removed from the user. The results can be seen in figure 5.3. Since we are mostly concerned with the values for each movie relative to the other movies from the same user, we also normalized the values, shown in figure 5.4.

In pursuit of better correlation and insight into the data, noisy data points or outliers were removed prior to the analysis. In order to do this, we held out each user profile, one by one, from the dataset. The R^2 of the remaining dataset was calculated and used to exclude the user with the highest R^2. We repeated this procedure until we had reached a certain correlation. In order to determine this target correlation, we plotted the correlation after removing different amounts of users. This plot can be seen in figure 5.5.

We plotted the TCAV score against the change in ratings when the R^2 had reached a value of 0.1 in figure 6.4 and 0.2 in figure 6.5. Both figures are found in appendix B. 38.5% and 53.8% of the users were removed to achieve a R^2 value of 0.1 and 0.2, respectively.

We plotted the position of the movie with the highest TCAV score in the importance list for several users. This can be seen in figure 5.6. This figure does not include the movies which either had a TCAV score of zero or which changed the ratings by zero.

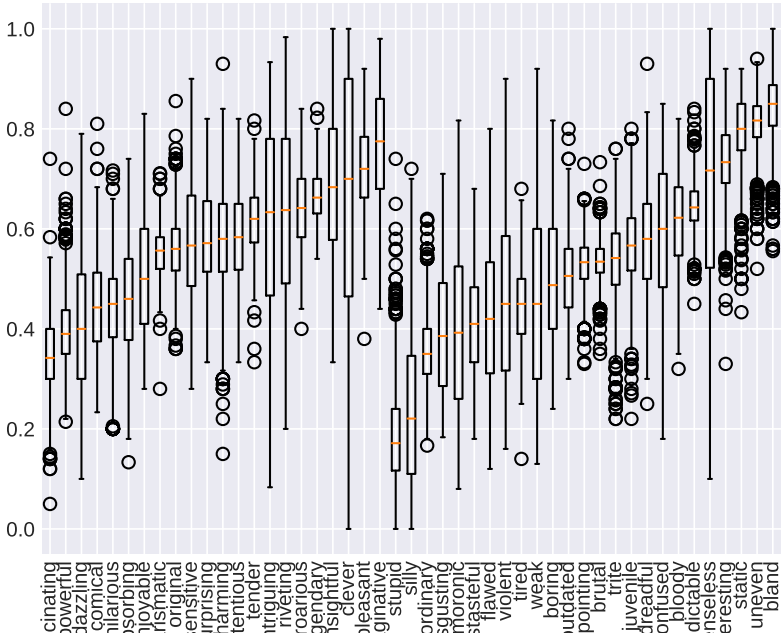


Figure 5.2: Distribution of the TCAV scores for positive and negative words

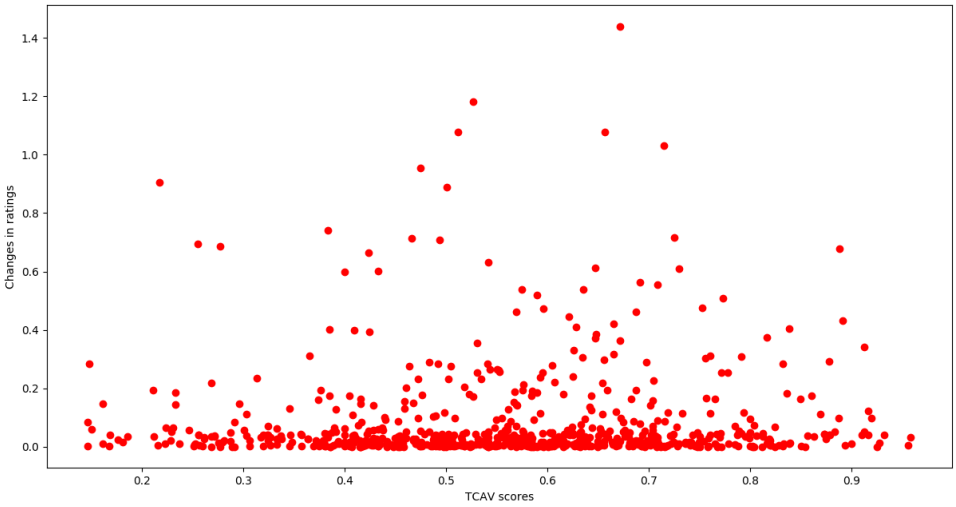


Figure 5.3: Change in tcav score plotted against change in predicted score

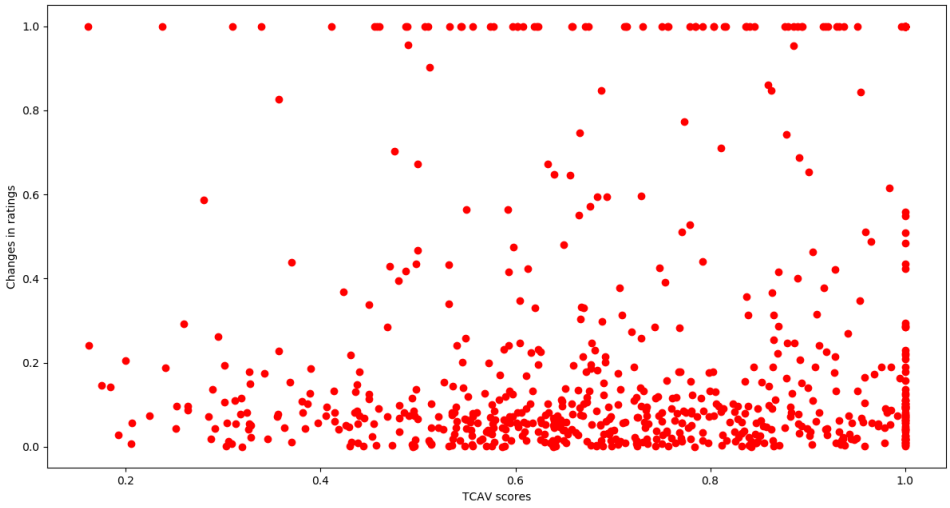


Figure 5.4: Normalized change in tcav score plotted against normalized change in predicted score

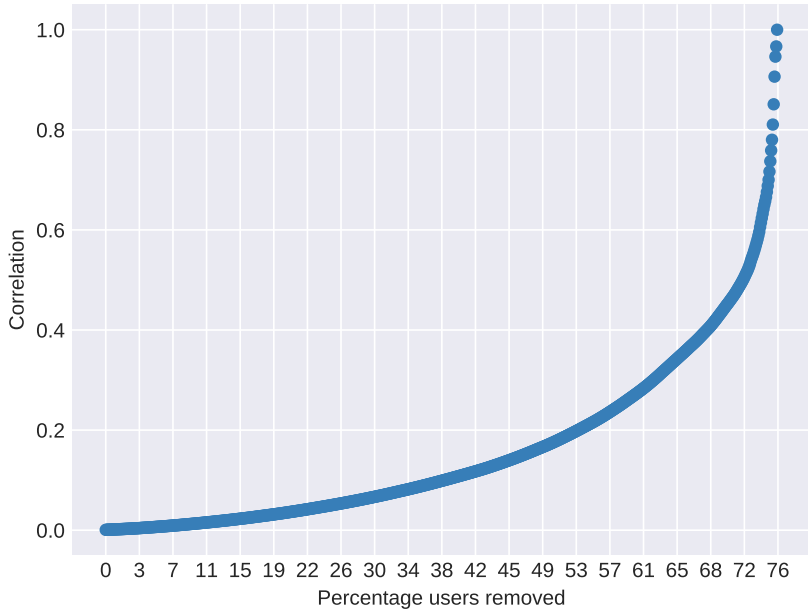


Figure 5.5: Correlation when removing a number of users

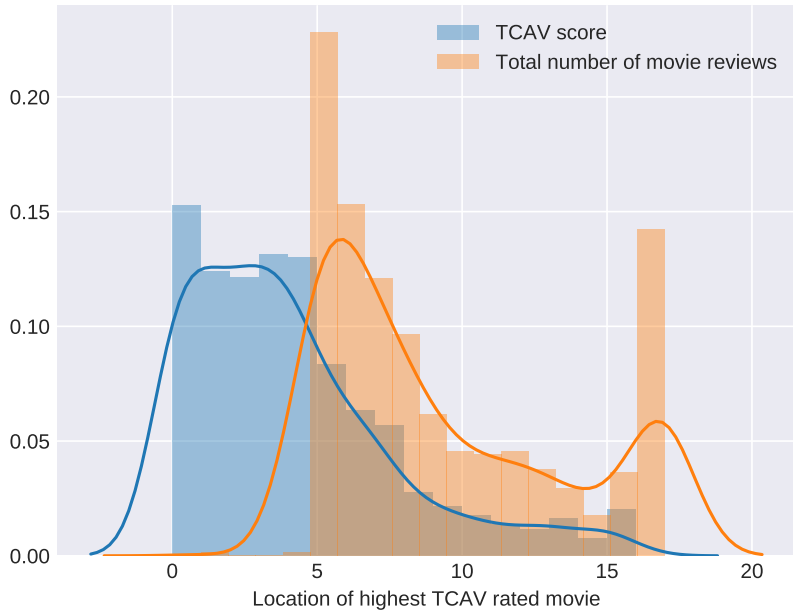


Figure 5.6: The position of the highest TCAV rated movie in the list of important movies for changes in rating

5.1.2 Component study

LRP

All $1.7 * 10^6$ reviews were processed with LRP on the CNN part of the model. There were $65 * 10^6$ triplets by the 100 filters on the corpus. $4.8 * 10^6$ of which were distinct triplets. Max pooling the triplets per filter, per document, reduced the number of triplets from $4.8 * 10^6$ to $1.5 * 10^6$. Despite our best efforts, we were unable to explain the CNN. The primary issue was the clustering of the word embeddings. Each of the filters had, on average, 14580 distinct triplets. Sorting these by activation yielded rare triplets which likely did not contribute much to the model accuracy. The clustering of these gave, in our eyes, many seemingly random words. Sorting the triplets by frequency, on the other hand, often led to triplets which did not belong together. This was evident by the low activation score the triplets had. Clustering these triplets gave few triplets, but again, these were also very little concise.

Inspecting the score when multiplying filter slot weights with all word embeddings gave similar results. Again, some words were ranked very high, but these were either very infrequent or absent from the data set. This is likely because the vocabulary of the reviews is a subset of the embedding vocabulary.

As seen in figure 5.7, there is some negative correlation between the first and last slot in the filters. The inspection of this effect is left for future work. A small sample of the data during this procedure can be found in table 6.1 in appendix A.

Figure 5.8 shows the RMSE for different ways of sorting and removing triplets from the input. All methods sort triplets by relevance score and keeps the highest scoring ones. *Relevance* is the default sorting method. Relevance scores can be negative; thus, we sort triplets by the absolute value as well. Finally, we select the highest scoring triplet, per filter, per document.

Figure 5.9 RMSE as a function of number of unique triplets kept. *Occlusion* replaces the words between triplets with the embedding of `<PAD/>`. *Wide occlusion* is similar, but it keeps quintuplets, i.e., including the neighboring words to a triplet, before replacing the dead space with the pad value. *Cut* keeps triplets and cuts out the words between triplets, before concatenating the remaining triplets.

Finally, figure 5.10 shows the RMSE as a function of triplets kept, when removing triplets by occlusion, sorting by max pool method.

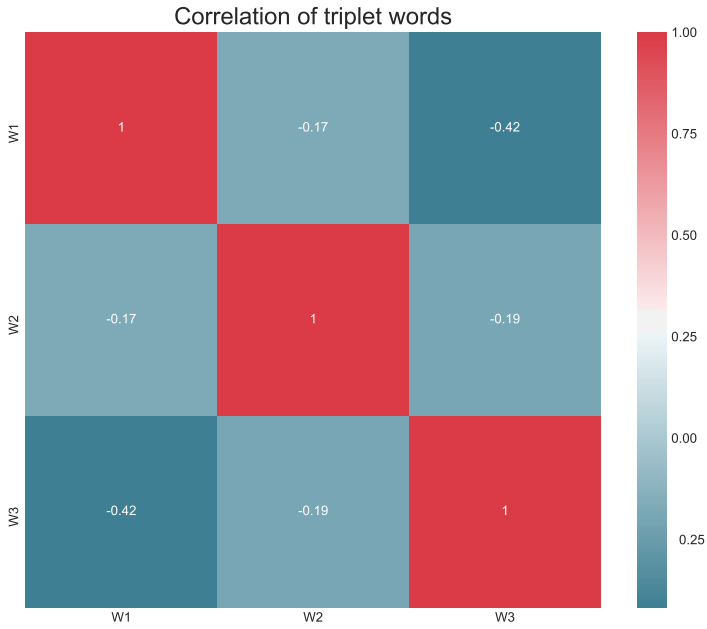


Figure 5.7: Correlation of triplet features. $W1 - W3$ is the relevance scores for slot 1 - 3.

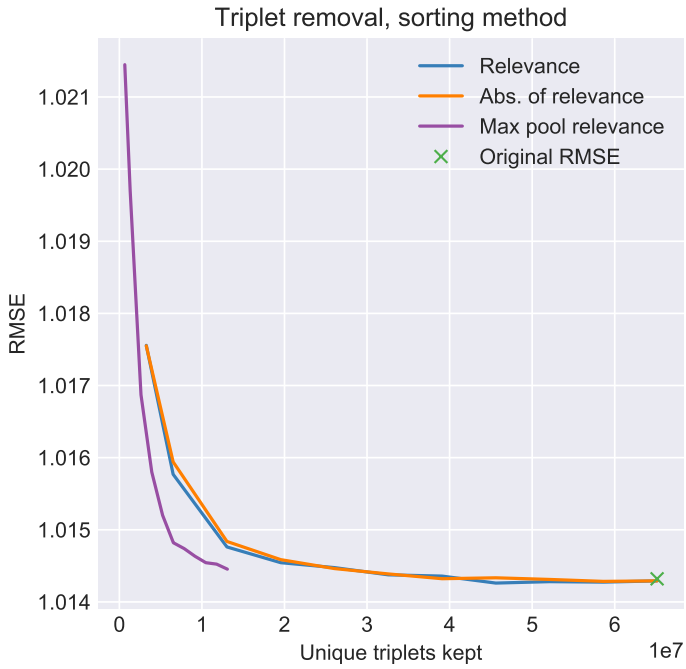


Figure 5.8: Change in RMSE for different triplet sorting methods

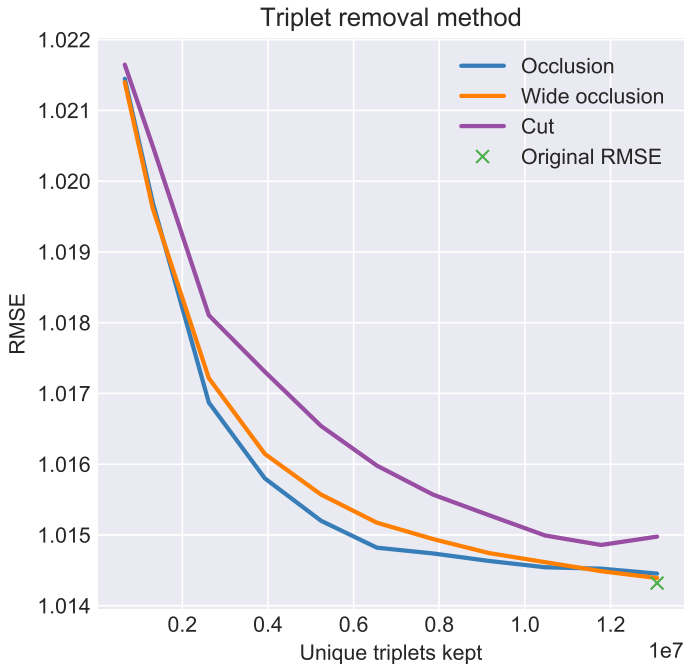


Figure 5.9: Change in RMSE for different triplet removal methods

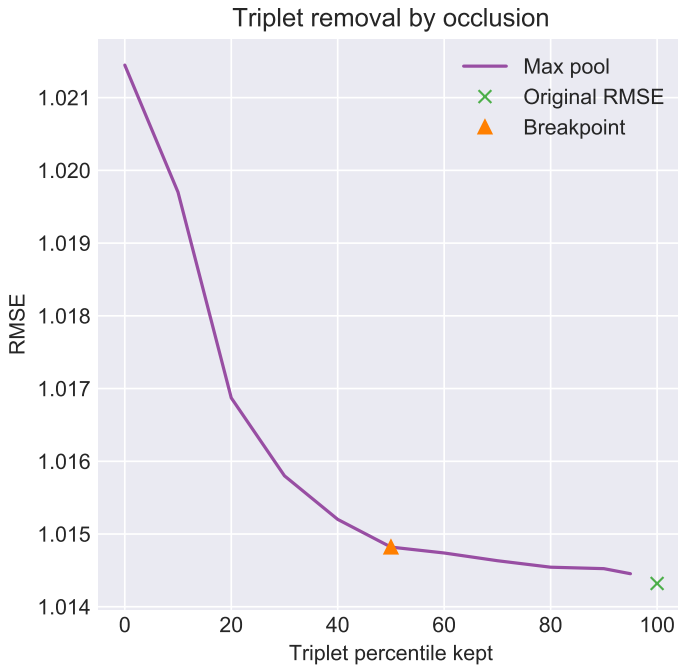


Figure 5.10: RMSE as a function of triplets kept. Any triplet not in the percentile is removed from the input data.

5.2 Discussion

Our approach is built on NARRE, which makes it dependent on a data set with the same input features. Reviews are the only interaction between users and items that this system supports, which is a drawback. Reviews might not be available or applicable to a specific recommender system and other interactions are not supported.

5.2.1 TCAV

Topics as concepts

We could not discern any meaningful patterns in the topic words we tested. An interesting property of the TCAV scores for each topic is the spread of the data. A wide spread indicates that the topic is contrasted, while a narrow spread means that all users are equally affected by the topic. If a small change in TCAV score results in a significant change in suspected interest, compared to the rest of the userbase, the measure might not be fit to use as a basis for explanations. The topics we would be most interested in is, therefore ”cast”, ”positive”, and ”predictable”.

Sentiment as concepts

The positive words found from the manual annotation had an interesting property: a wide distribution of TCAV scores compared to other topics. We believe that the method with TF-IDF, which was used for the other topics, captures less relevant information than the manual annotation. Intuitively this makes sense; the CNN does more than just counting the words in the reviews.

Positive/negative words as concepts

Generally speaking, the spread of each concept here is wider compared to the spread of the concepts in the ”Topics as concepts” approach. This could be explained by claiming that the model tries to find generally positive and negative reviews, but considers the writing style when doing so. If the language used in a review matches the language used by a user, the user will be more affected by the sentiment of the review.

Validating TCAV

Quantifying the correlation between the TCAV score and the changes in the ratings were done by training a linear classifier on the data, and evaluating the coefficient of correlation (R^2). If there had been a perfect correlation, this quantity would be 1. However, we do not expect a perfect correlation since TCAV is defined differently from merely the change in ratings. We hypothesize that the TCAV score and the change in ratings would both be a measure of importance, and as such, should be somewhat correlated. Ideally, what we would like to see in these plots are many points along the diagonal, which would indicate that there is a correlation between TCAV scores and changes in ratings. In these plots, there seems to be very little correlation.

There does not seem to be any apparent breakpoints in the plot showing the hypothesis fit at different numbers of users removed.

In the TCAV validation done until now, the focus has been on whether the ordering of the TCAV scores have correlated with the changes in the ratings if the movie had been removed. A different perspective is to consider the top TCAV rated movie and investigate what its importance to the user ratings are. This perspective is more relevant for explanations that are based on a single related movie instead of multiple movies.

We can see that the highest TCAV rated movie tend to be high in the importance list. This is promising for our hypothesis. Evaluating the TCAV score for a user is a quick way of estimating the most essential movie a user has seen, and this result hints that it could be accurate as well.

5.2.2 LRP

Sorting method

Sorting the triplets by relevance score or absolute value of relevance score is very similar, and we cannot assert the effects, or the lack thereof, of negative relevance scores. This is seen in figure 5.8. While the blue curve is slightly below the orange, the difference remains minimal. Reducing the triplets by max pooling, and then sorting the remaining triplets on their relevance scores, has a relatively low impact on the RMSE. We believe that this is because the model performs max pooling on the triplets which, in a way, sets a threshold for triplet relevance score. We choose the latter result for the remaining work.

When inspecting the effects of triplet removal by max pool sort in figure 5.10, we observe that the slope decreases rapidly until the 50th percentile, after which it tapers off. The breakpoint at the 50th percentile represents a small trade-off in performance for a considerable reduction of triplets. It should be noted that the max pooling operation has already reduced the triplet count by 30%, while still having a good performance before any thresholding. This is marked as the green X in figure 5.10.

Triplet removal method

The simple cut method has the highest RMSE, as seen in figure 5.9. We believe that new triplets are produced when doing this operation, which can be considered unnatural. The same reasoning applies to the wide occlusion method, but to a lesser extent. The occlusion method has the lowest RMSE, which disproves our theory of a negative impact of the padding value.

Interpretation

As with MF-based algorithms, the latent space is not necessarily interpretable. Embeddings are often latent spaces, which makes them difficult to interpret as well. This is an issue, as the filters themselves could be traced back to the latent space but little information would be gained. Similar words would be close in the latent space, and a filter slot would generate close values for these words. This also means that a filter slot could be an *average* of multiple words, which is counter-intuitive of words having a single definition.

The dimensionality of the word embeddings was an issue. We attempted to use mean shift clustering on each of the filter slots, as the original authors did, but the obtained clusters accounted for a meager percentage of triplets that were activated. These words were considered to be outliers in the data set. Therefore, we tried to assign custom weights to the words when performing the clustering. Such weights could be a function of relevance score, total activation count across all filters, and activation count within each filter number. This conflicts with mean shift, as it operates with its internal weights and is indifferent to prior assigned weights. The weights can be used with DBSCAN [77], where the weights would increase or decrease the minimum required points for a cluster to be defined as a core cluster. However, the results from DBSCAN was inconclusive as well. Some of the issue was related to the need for a hyperparameter *eps* which essentially decides how many clusters there are. The other issue was that the custom weight function must strike a balance between the importance of word relevance and word occurrence count. There were multiple clusters, but each cluster only consisted of one word. This could be because the assigned weight for each triplet was too high, favoring the case where each word appeared to be separate clusters. On the other hand, reducing the effect of word occurrence count resulted in similar clustering to that of mean shift.

Another issue was that around 40% of the distinct words for a filter slot were not present in the embedding. This is likely because of bias layers in the network, which essentially adds some value to each word. This causes LRP to assign relevance to the word, even though the word has a zero vector in the embedding space. A solution could be to pre-process the dataset, prior to the training, by removing all words not present in the embedding. Another solution would be to use a word embedding with support for out of vocabulary words, such as ELMo. This requires some work as different word embeddings employ different pre-processing and hyperparameters. The model would also have to be retrained.

We manually interpreted the triplets in the filters. This is a time-consuming task and would require maintenance over time if the model is to be continuously updated. It also introduces room for human error and biases.

The number of filters also affects interpretability. It is easier to perceive the information from a few triplets, rather than many triplets. One would have to choose a stringent threshold for deciding on how relevant a filter is. This threshold should strike a balance between ease of perception and accuracy of explanation, as these two are, in our case, trade-offs between each other.

Another issue is that the max-pooled output of all the filters can be related in the NCF-component of NARRE. This means that we would not be able to capture such dependencies by examining the CNN-part of NARRE solely. Any further explanations from the NCF could be neighbor-based, i.e., grouping filters together by inspecting their distances in the latent space of the NCF. In turn, this could lead to a clustering of filter triplets.

5.3 Research questions revisited

5.3.1 RQ1

We have seen different classical recommender systems and explanations from these in section 2.1. Some use item metadata as explanations. This metadata can be very interpretable but can be difficult to acquire. Other systems use a subset of user-item interactions as explanations. These do not rely on metadata, but explanations can be shallow.

We have observed that there is not a clear definition of explanations. Some authors define it as a justification of a recommendation which is presented to the user in a natural language. Others define it as interpretations or insights in the model, which are presented as raw values.

5.3.2 RQ2

With some overlap from RQ1, general interpretation techniques from ML can be applied in deep learning as well. However, the increasing complexity of neural nets requires new methods. This includes TCAV, LRP, and custom methods of inspection of different layers.

We have looked at general interpretation techniques of ML models in section 2.4.2. We divided our research into local and global explanations: the former is deeply rooted in the input data and its effect on the predictions. The latter is more general and can also be used to check model bias or correctness.

5.3.3 RQ3

We narrowed down the scope of this question by inspecting a single recommender system, NARRE, and a subset of its different parts. We applied TCAV to this model, with mixed results. It is dependent on well-defined concepts, but it shows some promise. Our analysis of the CNN was not as successful, yielding only insights into the troubles of interpreting filters and embeddings. We have also discussed the difficulties of interpreting latent spaces, which many recommender systems use.

5.3.4 RQ4

We use an offline metric to validate the output from our explanations. This metric measures the response on the prediction quality when removing certain input features. This lets us test different strategies when explaining or interpreting our results and methods.

Chapter 6

Conclusion and future work

6.1 Conclusion

We have seen that there are different approaches to explaining recommender systems and different limitations that must be considered. The theory in section 2.1.3 goes hand-in-hand with how models can be more (explicitly) interpretable. This literature provides useful guidance in pursuit of more explainable models. In this thesis, we have focused on methods which explain the architecture of the model. Instead of treating the model as a black box, we attempt to open the black box. We also review methods for black box models, which uses the relation between input and output to infer the behavior of the model. Ethical and legal issues become more pressing, not to mention the trust users have in recommender systems; the relevance of having adequately explainable models will increase.

We see from our work that it is possible to find explanations for methods which historically has been considered black boxes. Our attempts to explain a CNN component illustrates the difficulties one may face in the quest for interpretability of a deep learning model. Nevertheless, as the focus of the industry shifts from ”raw” measures such as RMSE to more user-experience oriented ones, we predict that there eventually will exist complex architectures that are inherently explainable. We introduce suggestions for concepts in the domain of review, which is a possible avenue to gain insight into recommender system that considers reviews.

6.2 Future work

Several things could be interesting to pursue in future work.

6.2.1 Improvements to the core recommender system

Several changes to the core recommender engine could be made to increase the accuracy or increase its explainability. The analysis in section 5.1.2 forms the basis for experimenting with different word embeddings. As discussed in section 2.2, replacing word2vec with BERT or ELMo would increase the accuracy of the model and improve our triplet analysis.

There is also the possibility of incorporating *Explainable matrix factorization* as described by Abdollahi and Nasraoui [48]. It should be noted that EMF does not provide explanations itself, but aims to increase the quality of neighborhood-based explanations.

We would continue the research of the CNN analysis by investigating the correlation between the triplet weights. We would approach the CNN more methodically, evaluating our assumptions and computational methods to ensure that our current results are correct. Finally, we would replicate the overall method on other deep learning NLP models with CNNs, to explore how CNNs work in general on NLP problems.

While unsuccessful with our analysis of the CNN, we would attempt to regularize the CNN. This could be done by inventing a metric which disfavors many distinct triplets or favors triplets with low semantic distance. This metric would then be included in the loss function for the network, similar to EMF as mentioned above.

Explaining the NCF component of the model is also something that could be done. It could build on some of the analysis from the CNN, providing neighborhood explanation in terms of triplets.

6.2.2 Explanations

Several desired properties of recommender systems were introduced in section 2.1.3 . The model performance on these properties could benefit from explanations. For example, increasing the unexpectedness of the system could decrease how much the users trust the system. The use of convincing explanations can regain this trust. Similarly, there is still some functionality required by GDPR, which is not yet possible in most recommender systems. If a user requests to be deleted from the system, the core recommender model would have to be retrained in order to remove any trace of the user entirely. Exploring this is an exciting area as well.

Clustering items can facilitate insight into the data. With TCAV, one could calculate a pair-wise distance metric between movies based on the TCAV score assigned to users to the movie-concept. This idea is touched upon in section 4.1.1.

The results from the TCAV approach suggests that using TF-IDF may not be an ideal method to find reviews that relate to the same topic. Alternatives to TF-IDF could be grouping reviews based on the similarity score given by a paraphrase detection model [66], or using a generative model to compose reviews on topics.

Bibliography

[1] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1181. URL https://www.aclweb.org/anthology/D14-1181.

[2] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, and Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). *arXiv preprint arXiv:1711.11279*, 2017.

[3] Yongfeng Zhang and Xu Chen. Explainable recommendation: A survey and new perspectives. *CoRR*, abs/1804.11192, 2018. URL http://arxiv.org/abs/1804.11192.

[4] Chong Chen, Min Zhang, Yiqun Liu, and Shaoping Ma. Neural attentional rating regression with review-level explanations. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 1583–1592, Republic and Canton of Geneva, Switzerland, 2018. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-5639-8. doi: 10.1145/3178876.3186070. URL https://doi.org/10.1145/3178876.3186070.

[5] Bin Hao, Min Zhang, Yunzhi Tan, Yiqun Liu, and Shaoping Ma. Are ratings always reliable? discover users' true feelings with textual reviews. In Min Zhang, Vincent Ng, Dongyan Zhao, Sujian Li, and Hongying Zan, editors, *Natural Language Processing and Chinese Computing*, pages 442–453, Cham, 2018. Springer International Publishing. ISBN 978-3-319-99495-6.

[6] Sushma Channamsetty and Michael D. Ekstrand. Recommender response to diversity and popularity bias in user profiles. In *Proceedings of the Thirtieth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2017, Marco Island, Florida, USA, May 22-24, 2017.*, pages 657–660, 2017. URL https://aaai.org/ocs/index.php/FLAIRS/FLAIRS17/paper/view/15524.

[7] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009. ISSN 0018-9162. doi: 10.1109/MC.2009.263. URL http://dx.doi.org/10.1109/MC.2009.263.

[8] Netflix. Netflix ab testing description. https://medium.com/netflix-techblog/interleaving-in-online-experiments-at-netflix-a04ee 2018. Accessed: 2018-11-01.

[9] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272, Dec 2008. doi: 10.1109/ICDM.2008.22.

[10] Komal Kapoor, Vikas Kumar, Loren Terveen, Joseph A. Konstan, and Paul Schrater. "i like to explore sometimes": Adapting to dynamic user novelty preferences. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, pages 19–26, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3692-5. doi: 10.1145/2792838.2800172. URL http://doi.acm.org/10.1145/2792838.2800172.

[11] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. Beyond accuracy: Evaluating recommender systems by coverage and serendipity. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 257–260, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-906-0. doi: 10.1145/1864708.1864761. URL http://doi.acm.org/10.1145/1864708.1864761.

[12] Panagiotis Adamopoulos and Alexander Tuzhilin. On unexpectedness in recommender systems: Or how to better expect the unexpected. *ACM Trans. Intell. Syst. Technol.*, 5(4):54:1–54:32, December 2014. ISSN 2157-6904. doi: 10.1145/2559952. URL http://doi.acm.org/10.1145/2559952.

[13] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. Improving recommendation lists through topic diversification. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 22–32, New York, NY, USA, 2005. ACM. ISBN 1-59593-046-9. doi: 10.1145/1060745.1060754. URL http://doi.acm.org/10.1145/1060745.1060754.

[14] Zachary Chase Lipton. The mythos of model interpretability. *CoRR*, abs/1606.03490, 2016. URL http://arxiv.org/abs/1606.03490.

[15] Been Kim, Elena Glassman, Briana Johnson, and Julie Shah. ibcm: Interactive bayesian case model empowering humans via intuitive interaction. In *CSAIL Technical Reports*, 2015.

[16] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 51:141–154, 04 2011. doi: 10.1016/j.dss.2010.12.003.

[17] Tomoko Murakami, Koichiro Mori, and Ryohei Orihara. Metrics for evaluating the serendipity of recommendation lists. In Ken Satoh, Akihiro Inokuchi, Katashi Nagao, and Takahiro Kawamura, editors, *New Frontiers in Artificial Intelligence*, pages 40–46, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-78197-4.

[18] Derek Bridge and Marius Kaminskas. Measuring surprise in recommender systems. In *Workshop on Recommender Systems Evaluation: Dimensions and Design (REDD 2014) at ACM RecSys 2014*, 2014.

[19] Gerlof Bouma. Normalized (pointwise) mutual information in collocation extraction. *Proceedings of the Biennial GSCL Conference 2009*, 01 2009.

[20] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22 (1):5–53, January 2004. ISSN 1046-8188. doi: 10.1145/963770.963772. URL http://doi.acm.org/10.1145/963770.963772.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[22] Helge Langseth, Keith Downing, and Eliezer de Souza da Silva. Tdt-76 - deep learning, høst 2018. https://www.idi.ntnu.no/emner/it3105/tdt76/index.php?spraak=norsk, 2018. Accessed: 2019-05-07.

[23] Geoffrey Hinton E., Alexander Krizhevsky, Ilya Sitskever, and Nitish Srivastva. System and method for addressing overfitting in a neural network, 2013. URL https://patents.google.com/patent/US20140180986A1/en. US Patent 20140180986A1.

[24] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing [review article]. *IEEE Computational Intelligence Magazine*, 13(3):55–75, Aug 2018. ISSN 1556-6048. doi: 10.1109/mci.2018.2840738. URL http://dx.doi.org/10.1109/MCI.2018.2840738.

[25] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey A. Dean. Computing numeric representations of words in a high-dimensional space, 2013. URL https://patents.google.com/patent/US9037464B1/en. US Patent US9037464B1.

[26] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[27] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

[28] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018. URL http://arxiv.org/abs/1802.05365.

[29] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

[30] Malvina Nissim, Rik van Noord, and Rob van der Goot. Fair is better than sensational:man is to doctor as woman is to doctor, 2019.

[31] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. Attentive collaborative filtering: Multimedia recommendation with item- and component-level attention. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 335–344, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5022-8. doi: 10.1145/3077136.3080797. URL http://doi.acm.org/10.1145/3077136.3080797.

[32] Wenpeng Yin, Hinrich Schütze, Bing Xiang, and Bowen Zhou. ABCNN: attention-based convolutional neural network for modeling sentence pairs. *CoRR*, abs/1512.05193, 2015. URL http://arxiv.org/abs/1512.05193.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

[34] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. *CoRR*, abs/1708.05031, 2017. URL http://arxiv.org/abs/1708.05031.

[35] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology Behind Search*. Addison-Wesley Publishing Company, USA, 2nd edition, 2008. ISBN 9780321416919.

[36] Peter Rousseeuw. Rousseeuw, p.j.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. comput. appl. math. 20, 53-65. *Journal of Computational and Applied Mathematics*, 20:53–65, 11 1987. doi: 10.1016/0377-0427(87)90125-7.

[37] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. *CoRR*, abs/1602.04938, 2016. URL http://arxiv.org/abs/1602.04938.

[38] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001. ISSN 00905364. URL http://www.jstor.org/stable/2699986.

[39] André Altmann, Laura Toloşi, Oliver Sander, and Thomas Lengauer. Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10): 1340–1347, 04 2010. ISSN 1367-4803. doi: 10.1093/bioinformatics/btq134. URL https://doi.org/10.1093/bioinformatics/btq134.

[40] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *CoRR*, abs/1312.6034, 2013.

[41] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10(7):1–46, 07 2015. doi: 10.1371/journal.pone.0130140. URL https://doi.org/10.1371/journal.pone.0130140.

[42] Philipp Wimmer. Methods for interpreting and understanding deep neural networks. https://hci.iwr.uni-heidelberg.de/system/files/private/downloads/1841142098/philipp_wimmer_eml2018_report.pdf, 2018. Accessed: 2019-05-04.

[43] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[44] Erik Bjørgen & Carl Johan Hambro. A survey of explainable movie recommendations. TDT4501: Computer Science, Specialization Project, 2018.

[45] Rose Catherine, Kathryn Mazaitis, Maxine Eskénazi, and William W. Cohen. Explainable entity-based recommendations with knowledge graphs. *CoRR*, abs/1707.05254, 2017. URL http://arxiv.org/abs/1707.05254.

[46] William Yang Wang, Kathryn Mazaitis, and William W. Cohen. Programming with personalized pagerank: A locally groundable first-order probabilistic logic. *CoRR*, abs/1305.2254, 2013. URL http://arxiv.org/abs/1305.2254.

[47] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. Ripple network: Propagating user preferences on the knowledge graph for recommender systems. *CoRR*, abs/1803.03467, 2018. URL http://arxiv.org/abs/1803.03467.

[48] Behnoush Abdollahi and Olfa Nasraoui. Explainable matrix factorization for collaborative filtering. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, pages 5–6, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-4144-8. doi: 10.1145/2872518.2889405. URL https://doi.org/10.1145/2872518.2889405.

[49] Xiangnan He, Tao Chen, Min-Yen Kan, and Xiao Chen. Trirank: Review-aware explainable recommendation by modeling aspects. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM

'15, pages 1661–1670, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3794-6. doi: 10.1145/2806416.2806504. URL http://doi.acm.org/10.1145/2806416.2806504.

[50] Yunfeng Hou, Ning Yang, Yi Wu, and Philip S. Yu. Explainable recommendation with fusion of aspect information. *World Wide Web*, Apr 2018. ISSN 1573-1413. doi: 10.1007/s11280-018-0558-1. URL https://doi.org/10.1007/s11280-018-0558-1.

[51] Hongning Wang, Yue Lu, and Chengxiang Zhai. Latent aspect rating analysis on review text data: A rating regression approach. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 783–792, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0055-1. doi: 10.1145/1835804.1835903. URL http://doi.acm.org/10.1145/1835804.1835903.

[52] Alon Jacovi, Oren Sar Shalom, and Yoav Goldberg. Understanding convolutional neural networks for text classification, 2018.

[53] DARPA. Darpa homepage. https://www.darpa.mil/, 2019.

[54] DARPA. Darpa xai homepage. https://www.darpa.mil/program/explainable-artificial-intelligence, 2019.

[55] David Gunning. Darpa explainable artificial intelligence program update. *DARPA/I2O*, 2017. URL https://www.darpa.mil/attachments/XAIProgramUpdate.pdf.

[56] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, December 2015. ISSN 2160-6455. doi: 10.1145/2827872. URL http://doi.acm.org/10.1145/2827872.

[57] Grouplens. Grouplens ab testing description. https://grouplens.org/datasets/movielens/, 2018. Accessed: 2018-11-08.

[58] Jesse Vig, Shilad Sen, and John Riedl. The tag genome: Encoding community knowledge to support novel interaction. *ACM Trans. Interact. Intell. Syst.*, 2(3): 13:1–13:44, September 2012. ISSN 2160-6455. doi: 10.1145/2362394.2362395. URL http://doi.acm.org/10.1145/2362394.2362395.

[59] NetflixPrize. Netflix prize. https://www.netflixprize.com/, 2018. Accessed: 2018-12-11.

[60] Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 507–517, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-4143-1. doi: 10.1145/2872427.2883037. URL https://doi.org/10.1145/2872427.2883037.

[61] Julian J. McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. Image-based recommendations on styles and substitutes. *CoRR*, abs/1506.04757, 2015. URL http://arxiv.org/abs/1506.04757.

[62] Julian McAuley. Recommender systems datasets. https://cseweb.ucsd.edu/~jmcauley/datasets.html#amazon_reviews, 2018.

[63] Mengting Wan and Julian McAuley. Item recommendation on monotonic behavior chains. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, pages 86–94, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5901-6. doi: 10.1145/3240323.3240369. URL http://doi.acm.org/10.1145/3240323.3240369.

[64] Chenwei Cai, Ruining He, and Julian McAuley. Spmc: Socially-aware personalized markov chains for sparse sequential recommendation. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, pages 1476–1482. AAAI Press, 2017. ISBN 978-0-9992411-0-3. URL http://dl.acm.org/citation.cfm?id=3172077.3172092.

[65] Anand Rajaraman, Jeffrey David Ullman, and Jure Lekovec. *Mining of Massive Datasets*, pages 1–17. Cambridge University Press, 2 edition, 2014. ISBN 9781107077232.

[66] Basant Agarwal, Heri Ramampiaro, Helge Langseth, and Massimiliano Ruocco. A deep network model for paraphrase detection in short text messages. *CoRR*, abs/1712.02820, 2017. URL http://arxiv.org/abs/1712.02820.

[67] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P11-1015.

[68] A. Hassan and A. Mahmood. Deep learning approach for sentiment analysis of short texts. In *2017 3rd International Conference on Control, Automation and Robotics (ICCAR)*, pages 705–710, April 2017. doi: 10.1109/ICCAR.2017.7942788.

[69] Kaggle. Amazon reviews for sentiment analysis. https://www.kaggle.com/bittlingmayer/amazonreviews/kernels, 2017. Accessed: 2019-03-03.

[70] stefanczarnecki. *Film Studies*. Simple Book Production, 2019.

[71] Been Kim. Tcav github repository. https://github.com/tensorflow/tcav, 2019.

[72] Chong Chen. Narre github repository. https://github.com/chenchongthu/NARRE, 2019.

[73] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[74] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

[75] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

[76] Maximilian Alber, Sebastian Lapuschkin, Philipp Seegerer, Miriam Hägele, Kristof T. Schütt, Grégoire Montavon, Wojciech Samek, Klaus-Robert Müller, Sven Dähne, and Pieter-Jan Kindermans. innvestigate neural networks!, 2018.

[77] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226–231. AAAI Press, 1996.

Appendix

A Datasets

Table 6.1: The format of amazon 5-core Movies and TV dataset

Item ID	User ID	Rating	Upvotes	Downvotes	Words	Review
14552	488	5	0	4	52	i ve been a fan of the series since i was a young boy personaly i don t consider the price of the boxset to high to keep me for me to placing my preorder i m sure this boxset will be a great offering from warner brothers and that i ll spend hours going through it

	filter_num	document	S1	S2	S3	W1	W2	W3	grouped_count	total_count	score
424113	38	1475413	yap	yap	yap	88.62	101.18	91.50	5	494	281.25
53036	57	135125	berkley	zachary	levi	61.21	104.56	112.31	1	100	278.00
113730	92	144255	yehuda	levi	levi	1.07	167.00	88.56	1	39	256.75
52216	92	66430	backwoods	ppl	aren	12.66	219.00	19.10	1	94	250.75
418507	40	1311620	puppy	zachary	levi	45.18	96.62	104.75	1	99	246.50

Figure 6.1: A sample of the data from the filter analysis

B TCAV plots

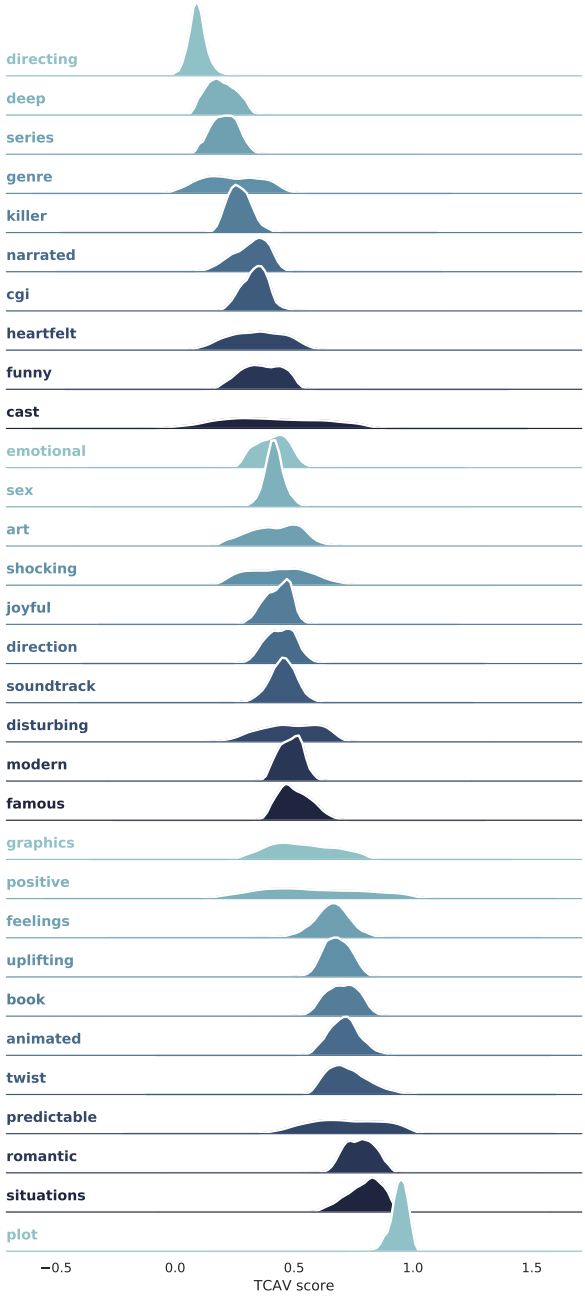


Figure 6.2: Density plots of TCAV score of various topics for several users

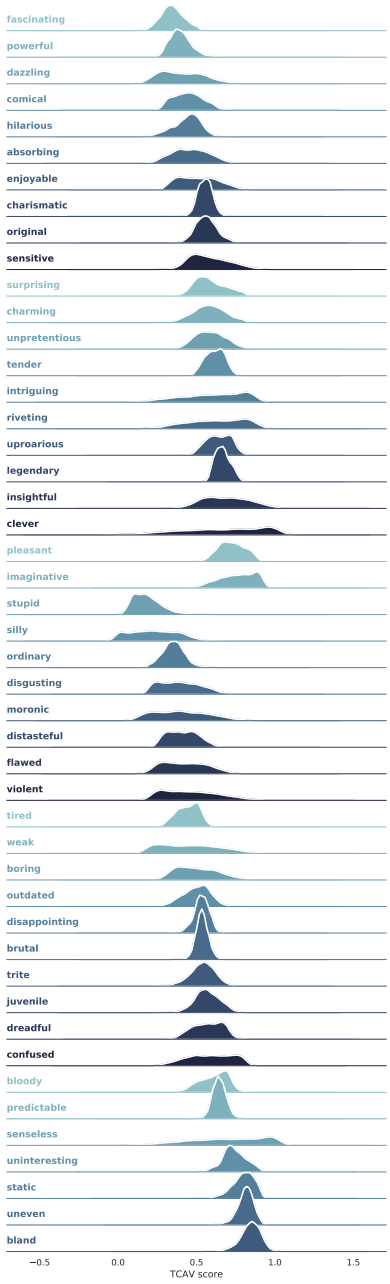


Figure 6.3: Density plots of TCAV score of positive and negative words for several users

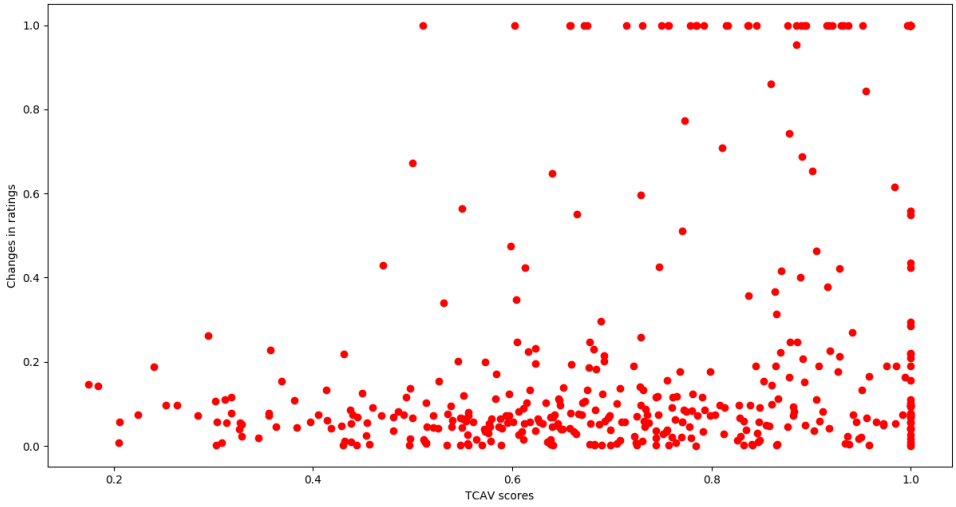


Figure 6.4: Normalized change in predicted score with R^2 of 0.1 as a function of normalized change in TCAV

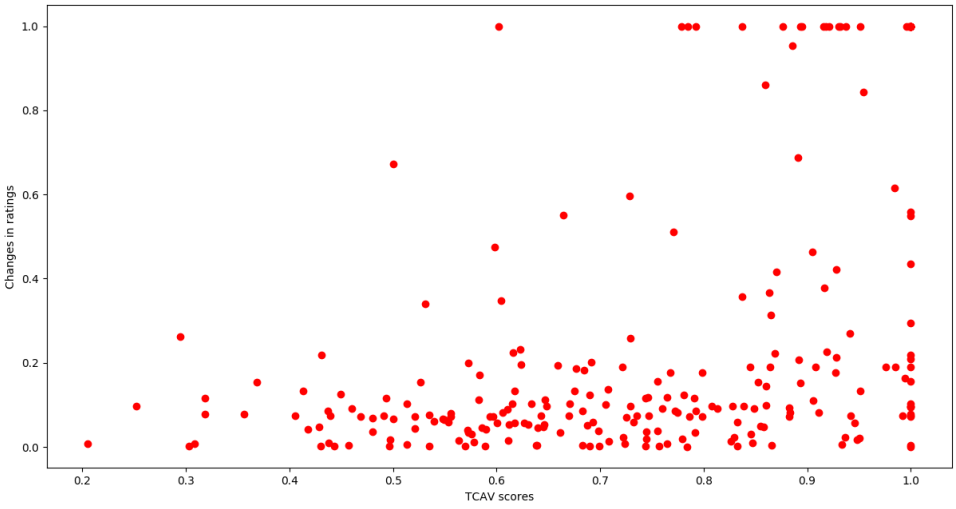
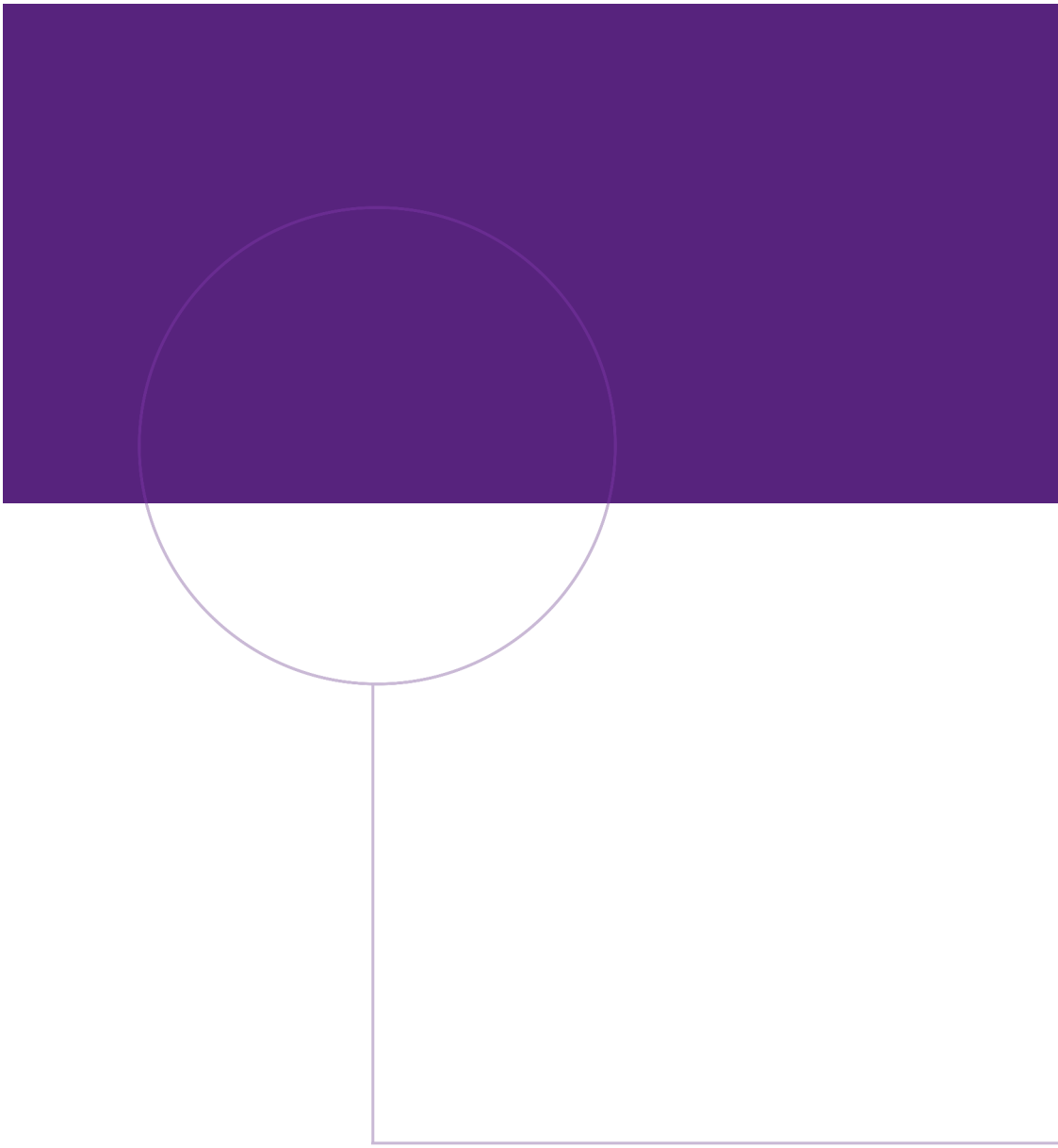


Figure 6.5: Normalized change in predicted score with R^2 of 0.2 as a function of normalized change in TCAV



NTNU
Norwegian University of
Science and Technology