Präferenzbasiertes Lernen von Produktempfehlungen auf Basis von Zalando Logs

Preference-based Learning of Recommendations from Large Scale Zalando Logs
Master-Thesis von Simon Holthausen aus Frankfurt am Main
Tag der Einreichung:

1. Gutachten: Prof. Fürnkranz
2. Gutachten: Christian Wirth

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Data and Knowledge Engineering

Präferenzbasiertes Lernen von Produktempfehlungen auf Basis von Zalando Logs
Preference-based Learning of Recommendations from Large Scale Zalando Logs

Vorgelegte Master-Thesis von Simon Holthausen aus Frankfurt am Main

1. Gutachten: Prof. Fürnkranz
2. Gutachten: Christian Wirth

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den July 16, 2015

(Simon Holthausen)

Abstract

This thesis utilises preference learning for recommender systems. Making recommendations can be seen as a ”learning to rank” problem, where all possible items are ranked and the first X items are included in a set of recommendations that is presented to the user. In this thesis, we propose rules that convert event data from click logs into pairwise preferences that are used for training. An example for said rules is the Sale rule, which creates the pairwise preference A > B if item A is bought by a user during a shopping session and item B is not. We also propose Conditional Functions, which are rules where the recommendation of the second item depends on the first recommendation, and the third recommendation depends on the second. The click logs used are provided by the shoes and clothing webstore Zalando. We train the rules with online learners, which can process big datasets in linear time. We analyse the rules regarding their performance in predicting views, carts and sales. The results show an improvement over the baseline, which is a popularity count of the items. Especially the results for predicting carts and sales are very promising.

Contents

List of Figures

List of Tables

List of Abbreviations

NDCG	Normalised Discounted Cumulative Gain
BPR	Bayesian Personalized Ranking
MOA	Massive Online Analysis
ROI	Return On Investment
AUC	Area Under the ROC-curve

1 Introduction and motivation

Since the first papers on collaborative filtering in the mid-90s, recommender systems became a more and more important research area. Both industry and academia are actively developing new approaches to recommender systems. The rise of the internet as an important means of consumer commerce has further contributed to this fact. Today, users watch movies online, buy clothing in webstores and are browsing the web for information. In all of these actions, recommender systems play an important role in reducing the overload of choices and narrowing them down to what the user might like. The problem of helping users with an overload of information is however still not suffeciently solved. A new way of tackling this problem is preference learning. Preference learning has been recognized as a particularly promising research direction for artificial intelligence. In 2010, Yahoo! hosted the Learning to Rank challenge [46].

In this thesis, an approach to recommender systems is proposed that uses preference learning. The approach does not use explicit ratings from users and also does not try to create explicit ratings, but learns from pairwise preferences, where one item is considered superior to another. These pairwise preferences are derived from implicit feedback, more specifically click logs of users. Deriving pairwise preferences more naturally reflects implicit feedback: If one wants to derive explicit ratings of items from implicit feedback, many potentially arbitrary assumptions have to made to create rules for creating such ratings. The dataset we are using is provided by Zalando, an online shoe and clothing store, and contains click logs of users. It is suitable for this thesis because it contains large amounts of real and relevant user events that can be converted into pairwise preferences.

The thesis is structered as following: First, the principles of preference learning and recommender systems are explained. Chapter 3 gives an overview of the related work. Chapter 4 introduces the approach proposed in this thesis and explains the rules that are used by the approach to create pairwise preferences from click logs. Chapter 5 consists of the evaluation. First, measures are introduced. Then, all methods outlined are tested on a small subset of the whole dataset. Once the best methods are identified, the results of those methods are validated on the whole dataset. Chapter 6 contains the conclusion, chapter 7 gives an outlook.

2 Principles

2.1 Preference Learning

Reasoning with preferences has been recognized as a promising research direction for artificial intelligence in the last years. In contrast to often very narrow and strict constraints, a preference can be considered as a relaxed constraint which can be violated to some degree, if necessary. This gives increased flexibility and is more directly associated with the thinking of a human being: It is easier to just prefer one item to another, than giving each item an absolute rating on a rating scale. Ultimately, a ranking can be determined out of many preferences. This task is at the core of preference learning and is called ”learning to rank”.

2.1.1 Preference Learning Tasks

The task of preference learning is as follows: There is some set of items for which preferences are known. One now has to learn a function that predicts preferences for new items, or for the same items in a different context. In most cases, one wants to derive a total order from the preferences, in which case we speak of a ranking problem, or the task of ”learning to rank”. There are three types of ranking problems, namely label ranking, instance ranking and object ranking, which are discussed below.

Label Ranking

In label ranking, one assumes an instance space X and a finite set of labels $Y = \{y_1, y_2, ..y_n\}$. From the instance space, one now wants to derive a total ordering $y_1 > y_2 > .. > y_n \in Sy$ of the labels, where Sy is the set of all total orderings of labels. In other words, one wants to learn a label ranker in the form of an $X \to Sy$ mapping. For example, one has a big dataset of clothing items with some features. Every item has one of the labels in $(brandnew, worn, old)$. Given as well is a set of preferences, for example item A is preferred to item B. From that set of preferences, one now can learn a total order of the labels which is $brandnew > worn > old$.

This learning scenario has a large number of applications. One example is meta-learning, where learning algorithms are ranked according to their suitability to a new dataset, based on the characteristics of this dataset [22]. Additionally, one can formulate the problem of classification and multilabel classification in terms of label preferences [19, 20, 21].

Instance Ranking

In this setting, an instance $x \in X$ belongs to one of the labels in $Y = \{y_1, y_2, .., y_n\}$. In constrast to label ranking, the ranking of labels is already given by a natural ordering $y_1 > y_2 > .. > y_n$. In contrast to classification, one does not want to learn a classifier, but a ranking function. At testing time, the labels are unknown and the ranking function has to rank the set of items according to their natural ordering.

For example, one wants to learn to correctly rank a set of clothing items which have one of the labels in $(brandnew, worn, old)$ where the natural ordering is $brandnew > worn > old$. At testing time, clothing items with unknown labels need to be ranked in such a way, that every item that should have the label ”brandnew” is above ”worn” and ”old”. The exact ordering is not important, but there is an error if an item that should have been labelled ”worn” is ranked above an item that should have been labelled ”brandnew”.

To measure the performance of a task, most approaches count the number of ranking errors. The scenario above would be counted as an error. For the case $|Y| = k = 2$, this amounts to the Area Under the ROC-curve (AUC) [25]. For multiple ordered rankings, one can use the concordance index or C-index used in statistics [26].

For the case of $|Y| = k = 2$, this problem is known as the bipartite ranking problem. For the case $k > 2$, it has been termed k-partite [23] or multipartite ranking [24].

Object Ranking

The task of object ranking is similar to instance ranking with the difference that there are no labels. The training information are only exemplary rankings or pairwise preferences in the form $z1 > z2$ meaning that z1 should be ranked higher than z2. An example is to rank the results of a query of a search engine [27, 28]. Training information here is implicitely provided by the user who clicks on the links in the result set he prefers. Another example is to rank clothing items in the context of a recommender system in such an order that the top three items are the ones the user most likely clicks on next. This example corresponds to our take on training recommender systems in this master thesis.

The performance of an object ranker can be measured in terms of a distance function or correlation measure. Because the rankingsets in the context of object ranking are often big, a measure is needed that puts more emphasis on the top of the rankings. An example is the Normalised Discounted Cumulative Gain (NDCG) [29,30], which does just that. In other applications, the bottom of the ranking might be ignored completely and only the ranking at the very top is important. This is the case in our approach to recommender systems where we only take into account the top three items for measurements.

2.1.2 Preference Learning Techniques

The three learning tasks in the previous section can be solved by similar techniques. Here, the two general approaches proposed in the literature are discussed. The first is learning some kind of utility function and assigning scores to the data, the second seeks to directly compare pairs of alternatives. In their introduction to preference learning, Fürnkranz and Hüllermeier also discuss model-based approaches and local aggregation of preferences [41].

Learning Utility Functions

Learning utility functions is an established approach to modelling preferences. Training data has to be in the form of instances together with their level of utility. How the level of utility is displayed, depends on the ranking task. In the setting of instance ranking, the training data consist of instances for which the utility scores are given directly as a number. An example is the rating history of a user on a movie site, where the user gives one to five stars to movies he has watched. In object and label ranking, the utitility function is typically in a more indirect form. Instead of scores, the learner is given constraints on the function. They are derived from comparative preference information, mostly in the form of pairwise preferences. For example, a user has two alternatives, item A and item B, and chooses item A. The function then learns to assign item A a higher utility score than B.

Learning Preference Relations

The idea of this approach is to directly learn binary preference relations from training data. Similar to learning utility functions for object and label ranking, the training data has the form of pairwise preferences. This time however, there is no utility function that later assigns a score to every instance, but - in theory - every combination of pairwise preferences is tested. Then a ranking is chosen, which is maximally consistent with the pairwise preferences. For example, if the items A, B and C are to rank, then (A>B, A>C, B>C), (A>B, A>C, C>B), ... etc. have to be tested. This leads to NP-hard problems [39]. Techniques such as simple voting have been proposed to deliver good approximations [40].

2.1.3 Applications of Preference Learning

Preference learning problems, ranking problems in particular, are part of many application areas. In information retrieval, the task of ranking the results or a query is such a task. One can apply preference learning methods to the problem of ranking the results. In particular, object ranking is used, where the set of results does not change, but its context does: Depending on the query, the documents have to be ranked differently. [31, 32] focus on this problem of transfering the ranking from one query to another. Another area is recommender systems, where the items are ranked according to some measure, for example similarity to the current item or similarity to the user's interests. The first x items then can be presented to the user as recommendations. We will discuss recommender systems in more detail in the next section.

2.2 Recommender Systems

Recommender systems emerged as a discipline mainly from information retrieval [15], with influences from cognitive science [16], approximation theory [17] and forecasting theories [18]. Due to their high demand and high applicability to the real world, much work has already been done in the field of recommender systems. We will briefly give an overview of different types of recommender systems and discuss some of their problems as well as proposed solutions.

2.2.1 Collaborative Filtering

This method identifies groups of people that share the same interests. If someone buys action movies, he will eventually be put into a cluster of people who share his interests. The recommender systems will then show him movies, people with similar interest did watch. This means Collaborative Filtering is based on the assumption that people who agreed in the past will agree in the future, and that they will like similar kinds of items as they did in the past. An example of this kind of recommender systems is the section ”Recommendations for you” on the landing pages of most shops when the user is logged in. Another example is Amazon’s recommendations-system, which uses not a user-item-matrix, but a item-item-matrix, to compute its ”people who bought x, also bought y”-reommendations.

Collaborative Filtering Approaches often suffer from a few problems:
New user problem: In order to construct clusters, the system needs a lot of existing users with ratings. If a new user is introduced, it takes some time to accurately identify the cluster the user belongs to to make accurate recommendations.
New item problem: The new item problem is essentially the same as the new user problem. Items need to be rated by enough users in order to get recommended. For both the new item and new user problem, hybrid systems offer solutions.
Sparsity: Only few users provide ratings for items, which are needed to learn. Due to the high amount of items in major e-commerce sites, only very few items are rated often enough to make reliable recommendations.
Grey sheep: If a user does not fit into a cluster, because he likes too many different things, the recommendation system cannot make accurate recommendations.

2.2.2 Content Based Systems

While Collaborative Filtering ignores the current item the user looks at, Content Bases Systems ignore the group of people a user belongs to and only focuses on the user itself. First a user profile is built from the history of items the user looked at and bought. It then finds items that are similar to the interest of the user and are put into the set of recommendations. The assumption of Content Based Systems is that the user wants to see ”more of the same”.

Like Collaborative Filtering, Content Based Systems suffer from a few problems:
Limited content analysis: Content Based Systems are inherently limited by the level of content analysis. Items need to be represented with a set of attributes. These are either extracted from text and for example converted into TF-IDF vectors, or inserted ”by hand”. If the attributes are insufficient, the system performs worse.
New user problem: If a new user is introduced to the system, the system has to learn about the user’s preferences in order to make accurate recommendations.
Overspecialization: If the taste of the user changes or he wants to see something ”out of the ordinary”, the recommender system cannot react quickly enough. For example, a person with no past history with Greek cuisine would not receive a recommendation even for the greatest Greek restaurant. A possible solution is to introduce randomness as described in the context of information filtering [1]. Another problem is that the user could get recommendations that are too similar to the current item. For example, the user reads a news article and then receives recommendations about the same news, only from other sources. A possible solution is to not only filter out items that are too different from the user’s taste, but also too similar to items seen before [2].

2.2.3 Hybrid Systems

Most Recommender Systems in use combine the methods above and tailor them to their needs or enhance them with additional features. Several papers [12, 13, 14, 7] empirically compare the performance of hybrid with collaborative and content-based methods and conclude that the hybrid methods can provide more accurate recommendations. Gediminas Adomavicius and Alexander Tuzhilin [3] separate the different ways to combine content-based and collaborative methods into a hybrid recommender system into four classes:

1. Combining separate recommenders: This method uses separate collaborative and content-based systems. They make independent recommendations. Those are then aggregated in some way to choose a final recommendation. Existing approaches use a linear combination of ratings [4], a voting scheme [5] or make use of so-called quality metrics, for example a confidence-level [6].

2. Adding content-based characteristics to collaborative models: This method tries to diminish problems of collaborative models by enhancing them with content-based characteristics. For example the ”collaboration via content” approach, described in [5], is based on traditional collaborative techiques, but maintains content-based profiles for each user, which are used to calculate the similarity between two users. This way, they overcome the sparsity-problem. They also can directly recommend an item if it ranks highly against the user’s profile and do not have to rely on ratings of users with similar interests. This way, they diminish the effect of the new item problem.

3. Adding collaborative characteristics to content-based models: This method employs collaborative characteristics on top of content-based models. The most popular approach in this category is to use dimensionality reduction techniques on groups of content-based profiles. [7] uses latent semantic indexing, where user profiles are represented by term vectors. The reduction of dimensionality often leads to performance improvements.

4. Developing a single unifying recommendation model: This method became more popular in recent years. Instead of putting characteristics of one model on top of the other, they are inherently mixed. [8] use features of users and items in a single rule-based classifier. [9] and [10] unify content-based and collaborative recommendations with probalistic semantic analysis. Others use Markov chain Monte Carlo methods for parameter estimation and prediction [11].

Summarising, there is no best approach to recommender systems: It highly depends on the environment the system is used in. Websites always have to tailor recommender systems to their needs.

3 Related Work

The approach to learning recommendations proposed in this thesis is based on the idea of object ranking presented in the previous chapter. The idea of learning from pairwise preferences was first formalized by Tesauro under the name comparison training [42]. A symmetric neural network was used with representations of two states and a signal, which indicates which of the states is preferable.
The authors of [43] propose Latent Pairwise Preference Learning for recommendations with implicit feedback derived from click logs. Their approach models both the latent variables of group structure of users and the pairwise preferences.

RankNet is a different approach to object ranking, which was introduced in the context of information retrieval [44]. For each instance a label is assigned to the goodness of the instance regarding the query, and a 1, 0 or -1 is assigned to a pair of instances if one is either superior to the other or they are equal. The model is trained via gradient descent. RankNet is optimising for the number of pairwise errors, but it does not match well with some other information retrieval measures like NDCG [29, 30].
To improve this, LambdaRank was proposed, which more directly tries to learn the NDCG [45]. LambdaRank is considered as one of the state-of-the-art-rankers, an optimised version called LambdaMART was used as part of the winner of the Yahoo! Learning to Rank challenge 2010 [46]. Further improvements were made by Szummer and Yilmaz with the proposition of a semi-supervised learning to rank algorithm. Their ranker, called SSLambdaRank, works with both labelled data in the form of pairwise preferences or absolute labels, and unlabelled data [47].

Bayesian Personalized Ranking (BPR) [48] is a comparably recent Collaborative Filtering method designed to deal with implicit-only feedback. It directly aims to optimise a ranking criteria, which is why it also can be considered as a learning to rank algorithm. The input consists of a user-item rating matrix, where a ”1” captures a positive interaction of a user with an item. The pairwise preferences derived from the implicit ratings are used to optimise an AUC ranking criterion. One limitation of the basic BPR model is that it only supports discrete, unary feedback. Some actions of users however might indicate stronger interest in an item than others, for example a buy event might be stronger than a view event.
To improve this, Lukas Lerche and Dietmar Jannach propose the extension BPR++ [49], which can take various forms of interactions and their importance into account. First, additional pairwise preferences are derived from the data depending on the importance of events, for example cart or sale events. During the optimisation phase, a certain amount of samples is drawn from these additional preferences to introduce bias.

The approach presented in this thesis combines the mixed approach of hybrid recommender systems with preference learning. The approach utilises click logs of users which show what items the user clicked on and what he did with them, for example view an item or put it into the shopping basket. The click logs are divided into sessions, which can be interpreted as a shopping trip. We create training examples from these past user sessions to train a model. We then use the model to recommend items to the user that other users with a similar session history have clicked on or bought. The assumption is that people who agreed in the past will agree in the future.

In training phase, pairwise preferences are created from click logs. For example, if a user looks at item A and then at item B, we create the pairwise preference B>A, in other words B is preferred to A. The assumption is that if the user liked item A, he would have carted it afterwards and not look at another item, therefore he prefers item B. All of the possible rules for the creation of pairwise preferences are discussed in section 4.1 and 4.2. A utility function is trained from the pairwise preferences. The concept of utility functions is explained in section 2.1.2.

When making new recommendations, the model's utility function is used to predict if a specific item is likely to be preferred to the item currently looked at. Let us assume the user is currently looking at item A and the system has to decide whether to recommend item B, C or D to him. It now uses the model trained and puts in the possible preference pairs A<B, A<C and A<D. The item that is most likely preferred over item A, i.e. is assigned the highest score from the utility function, is presented to the user.

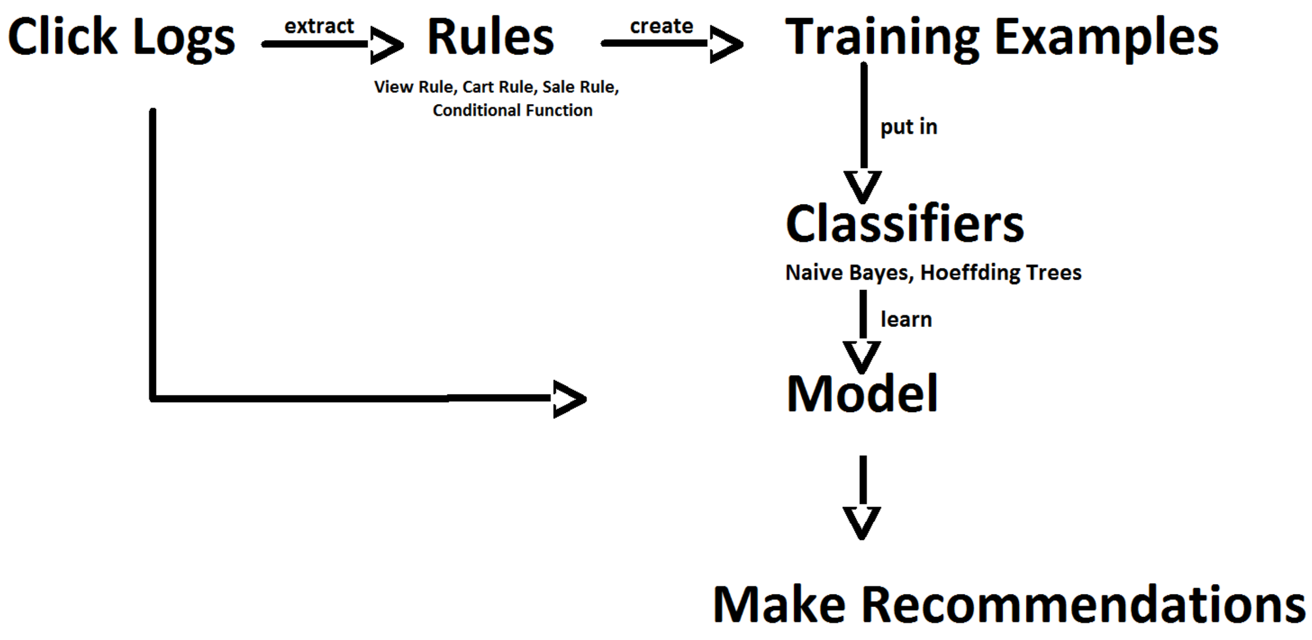


Figure 4.1: This flow chart shows an overview of the different steps taken in this thesis to make recommendations

The dataset is the Zalando Recommendations Dataset Version 2014. Zalando is an online shopping store that focuses mainly on shoes and clothing. The dataset consists of click logs of users. They are divided into sessions. Each session represents the continious interaction with the Zalando website over the course of several minutes. Every event is associated with several attributes, including the features of the item the user interacted with and the form of interaction. For training the model, the focus lies on three of the possible forms of interaction: View, cart, sale. A view event is the user viewing an item, a cart event represents the user putting the item into the shopping basket, and a sale event means the user bought an item. These three types of events are used to create rules for preference-pair-creation.

4.1 Rules for Preference-pair-creation

Below each rule is explained that is used for creating pairwise preferences.

4.1.1 Popularity Count Rule

This rule is only based on the popularity of items. For each item, the number of views, carts and sales is counted. For example if an item was viewed 10 times, carted 4 times and sold 1 time, it has a popularity count of 10+4+1=15. The more popular an item is, the higher it is ranked. This rule doesn't really need a training phase, just a lookup-table with items and their popularity count.

4.1.2 View Rule

This rule is based on the sequence of views a user makes in a session. If a user views item A and then views item B, then a pairwise preference is created where item A is inferior and item B is superior. The assumption is that if the user liked item A, he would have carted it afterwards and not look at another item, therefore he prefers item B. Other types of events like "Cart" or "Sale" are ignored. The figure below shows an example.

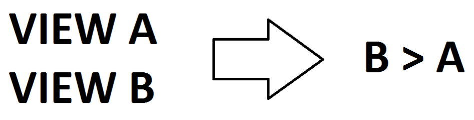


Figure 4.2: Example for View rule. User clicks on item A, then on B

4.1.3 Cart Rule

This rule is based on the cart events of a session. All items the user viewed are collected in sequence. If a cart event occurs, the item carted is superior to all items viewed up to that point. The item carted is removed from the set of viewed items before making the pairwise preferences. The assumption is that the user prefers items that he put in his shopping basket over items he did not put into it. The figure below shows an example.

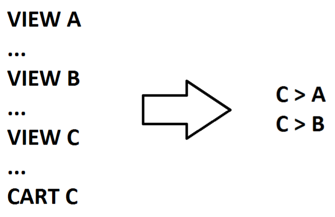


Figure 4.3: Example of applying the cart rule to the click log

4.1.4 Sale Rule

This rule is based on the sale events of a session. All items the user viewed are collected in sequence. If a sale event occurs, the item sold is superior to all items viewed up to that point. The item sold is removed from the set of viewed items before making the pairwise preferences. The assumption is that the user prefers items that he bought over items he did not buy. The figure below shows an example.

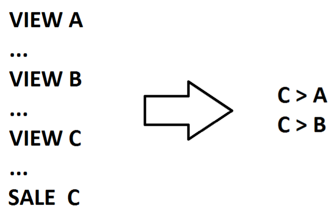


Figure 4.4: Example of applying the sale rule to the click log

4.2 Conditional Functions

Conditional Functions are combinations of rules extended with an additional feature vector of an item. This item is the condition, under which the rule is created. Unconditional rules like the ones explained above, for example the View rule or Cart rule, are of the form (notpreferredItem,preferredItem). Conditional rules are of the form (notpreferredItem,preferredItem|conditionalItem), which in the end is a feature vector of the form (notpreferredItem,preferredItem,conditionalItem). Conditional Functions consist of multiple conditional rules. We will first introduce the conditional rules, and then introduce their combinations with other rules which lead to Conditional Functions.

4.2.1 Conditional Cart

This rule first gathers preferences like the Cart rule explained in section 4.1.3: It gathers all carts of a session and creates pairwise preferences, where each cart is preferred to all items of the view events which did not get carted.
In the next step, carted items not included in sale events are discarded. Only those items remain which got carted but not bought. We now have a set of feature vectors of only-carted, and a set of carted-then-sold item preferences. We now form the cross product of those sets, where each feature vector of a preference in the set of only-carted items is extended with the feature vector of each item sold. The item sold is the conditional item. Additionally, the set of feature vectors of carted-then-sold item preferences is cross-multiplied with itself.

The reasoning behind this is conservative thinking: If we recommend an item, and this recommendation is wrong, meaning the user does not click on it, we want to recommend the next best item, and that is an item that is likely carted, but enhanced with our information about the item the user does not want. How we construct training examples for this rule is explained above. For making new recommendations, we use the result of another recommendation made earlier, and use the item obtained for this rule. The reasoning behind this is ”If he does not click on this recommendation, what will he likely click on then?”.

Example:
Item preferences only-carted: (A,B), (C,B)
Item preferences carted-then-sold: (A,D), (A,E), (B,D), (B,E)
Resulting item preferences: (A,B|D), (A,B|E), (C,B|D), (C,B|E), (A,D|E), (B,D|E), (A,E|D), (B,E|D)

4.2.2 Conditional Cart No Sale

This rule is the same as the rule above, but we leave out the cross product of the carted-then-sold item preferences with itself. The reason for this is that the cross product of carted-then-sold items may introduce too many contradicting training examples, since every item in that set is inferior as well as superior to each other item once.

Example:
Item preferences only-carted: (A,B), (C,B)
Item preferences carted-then-sold: (A,D), (A,E), (B,D), (B,E)
Resulting item preferences: (A,B|D), (A,B|E), (C,B|D), (C,B|E)

4.2.3 Conditional View

This rule gathers preferences like the View rule explained in section 4.1.2 does: It gathers all views of a session where the preferred item is the item viewed after the current item. In the next step, all viewed items not included in cart or sale events are put into a different set. We now have a set of only-viewed and viewed-later-carted itempreferences. Cross products are then made in the same way as explained above in the section Conditional Cart No Sale.

The reasoning behind this is conservative thinking: If we recommend an item, and this recommendation is wrong, meaning the user does not click on it, we want to recommend an item, that maybe doesn't get carted, but at least viewed. This recommendation is enhanced with the item the user does not want. For making new recommendations, we use the result of another recommendations made earlier, and use the item obtained for this rule. The reasoning behind this is again "If he does not click on this recommendation, what will he likely click on then?".

4.2.4 Conditional View No Cart/Sale

This rule is the same as the rule above, but we leave out the cross product of the viewed-then-carted item preferences with itself. The reason for this is that the cross product of viewed-then-carted items may introduce too many contradicting training examples, since every item in that set is inferior as well as superior to each other item once.

4.2.5 Conditional View No Cart/Sale Popularity Count Subset

This rule is the same as the rule above. But we choose only those pairwise-preferences as training examples, where a view following another view is in the top three of recommendations the Popularity Count rule would make. The reason behind this is that all training examples are filtered out that we cannot make use of later: Only training examples of item pairs are included that are in the direct neighbourhood of each other. The concept of neighbouring items is introduced in the section "Making Recommendations".

Positive example:
View A
Popularity Count recommendations: B, C, D
View C
→ used for training

Negative example:
View A
Popularity Count recommendations: B, C, D
View E
→ not used for training

4.2.6 Conditional Functions

We now introduce combinations of the rules above and the reasoning behind those combinations.
The Conditional Functions are straightforward: We first make a recommendation using a simple rule. We recommend an item, that is most likely sold later, so if the user clicks on it, there is a higher chance he will buy it later. Which rule is best to recommend such an item is validated later on. If the user does not click on it, we attempt to more conservatively recommend an item, the user will at least likely cart. If this item is not clicked on, either, we get more conservative and recommend an item the user is at least likely to view.

Example:
Currently Viewed Item: A
Likely sold: for each possible preference: Sale_Rule(A,possiblePreference) → Recommend B
Likely carted: for each possible preference: Cart_No_Sale_Rule(A,possiblePreference|B) → Recommend C
Likedly viewed: for each possible preference: View_No_CartSale_Rule(A,possiblePreference|C) → Recommend D
→ Recommended items: B, C, D

For each of the three recommendations made, we use different rules, their combinations make up the Conditional Functions. Following rules are used:

Likely sold: Top rule (or baseline) for predicting sales, which is determined later in tests.
Likely carted: Conditional Cart, Conditional Cart No Sale
Likely viewed: Conditional View, Conditional View No Cart Sale, Conditional View No Cart Sale Popularity Count Subset

This leads to 1 * 2 * 3 = 6 combinations to be tested.

4.3 Random Projection on Rules

After the rules and Conditional Functions above are evaluated, the best rules are identified and random projection is used on them to see if the results can be further improved.

Random projection is a graphical technique used to reduce the dimensionality of a set of points. The main idea behind random projection is the Johnson-Lindenstrauss lemma [33]. It states that if points in a vector space are in a sufficiently high dimension and are projected on a suitably high dimensional space, the distances between points remains almost preserved. To do such a projection, a random matrix is generated that projects the points to the lower dimension. Random projection methods are known for their simplicity, better computing time compared to methods like Principal Component Analysis and being less error prone [34].

We use a modified weka-implementation of random projection. Weka is a collection of machine learning algorithms for data mining tasks [35]. First evaluations on the small test set introduced in section 5.3.1 showed that a dimensionality reduction to 300 using Sparse1 as the generator for the random matrix produces the best results.

4.4 Classifiers

Two classifiers are used to train the models, namely Naive Bayes and Hoeffding Trees. For both, an implementation of the classifiers in the framework Massive Online Analysis (MOA) is used. MOA is a framework for online-learning [36]. Online-learning has the advantage that during training, every instance is only looked at once. This makes online-learners perform very well regarding computing time, which is especially important in this case when training on the big dataset.

Naive Bayes

The Naive Bayes classifier is a simple stochastic classifier. It applies Bayes' theorem to machine learning. Bayes' theorem states that $posterior = \frac{prior \times likelihood}{evidence}$

An instance is to be classified, represented by a vector $x = (x_1, ..., x_n)$, representing n features. From training we know $p(x|C)$, the likelihood of a variable given the class, as well as the prior probability $p(C)$ of the class. We now compute for each class the posterior probability $p(C|x) = \frac{p(C) \times p(x|C)}{p(x)}$. The class with the highest probability is chosen. This is the so-called decision rule which picks the hypothesis that is most probable, also known as the maximum a posteriori: $y = \arg\max_k p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$.

Naive Bayes assumes that all features are statistically independent, for example that the cost of a clothing item does not depend on its size. In reality, this is almost never the case - still, Naive Bayes works well in some areas like spam mail detection [37].

Hoeffding Trees

Hoeffding Trees are specifically designed for online-learning. They were proposed by Domingos and Hulten [38], who also called them Very Fast Decision Trees. It is an incremental, anytime decision tree induction algorithm that is capable of learning from massive data streams. Hoeffding Trees only need to observe every instance once. The assumption made is that the distribution of the data does not change over time. This way, Hoeffding Trees exploit the fact that a small sample can often be enough to choose an optimal splitting attribute. This is mathematically supported by the Hoeffding bound, which quantifies how many examples of instances are needed to achieve a certain level of confidence in the goodness of attributes.
In comparison to other incremental decision tree learners, Hoeffding Trees have sound guarantees on performance. With the Hoeffding bound it can be shown that the output of the learner is asymptotically nearly identical to that of a non-incremental learner.

4.5 Features used

We distinguish the features into the following categories:
Item Features: The features of an item, for example its colour. The simplest approach only uses these.
History Features: These features are the itemfeatures (or a subset of those) of the last n view events of the current usersession. Let us assume the history length is 2. At the start of the session, the history features are missing, because there is no history yet. After three view events, the features consist of the first and second item the user looked at.
Dependant Features: These features are used by the Conditional Functions. They are discussed in more detail in section 4.2.

4.6 Making Recommendations

In the sections above, steps to train the model are introduced. Once the model is trained, it is used to make recommendations in the following way: The item the user currently views is the not-preferred item. A set of items is then tried out as the preferred item. The model ranks the items that are tried out as preferred items. For example, let us assume the user looks at item A. Items B, C and D are tried out as preferred items, meaning the model determines $score(A, B)$, $score(A, C)$ and $score(A, D)$ and ranks the items accordingly. The more an item is preferred to the current item, the higher the score.

Ideally we would rank all items that are available each time a user looks at an item. This is not possible in a reasonable amount of time however: The dataset contains about 500,000 different items, ranking them all would take several seconds which is too long, since the user should see the recommendations immediately when the site is loaded. Hence we need to narrow down our test cases. In order to do so, we set up a graph of items beforehand. An edge from one item to another is drawn if the items differ in exactly one feature. We only have about 10 features and many of those features are categoryfeatures, hence this is reasonable. Now, when we make a recommendation, the item the user currently looks at is the starting point. Our algorithm then explores all its neighbours, only those items are used for ranking. The top three items to recommend are chosen out of this set. If we only make on iteration of the algorithm, we return those three recommendations. We also can make more than one iteration, in that case these top three recommendations are used as starting points for another exploration of the graph.

The following algorithm summarises the approach:

```
CurrI = item currently looked at
X = set of items
R = empty ranking of items

X += CurrI
For n times do:
    I = empty itemset
    For x in X:
        I += get items differing to x by one feature
    For each i in I:
        R += score(cI,i) (if not already present in R)
    X = top 3 of R

Output X
```

Notice that we do not clear the ranking after each iteration of the most outer loop. This means that if the best items are already found in the first iteration, they will stay at the top.

This algorithm implies that most recommended items should be in the neighbourhood of the item currently looked at. Indeed, in the dataset used, over 40 percent of all items users look at next are found after one iteration of the algorithm. More maximum possible scores of various measurements for iteration depth 1 and 2 are presented in section 5.4.1 and 5.5.1.

5 Evaluation

This chapter contains the evaluation. We first introduce measures for comparing the results. We then shortly introduce the two testsets on which the evaluation is made. Last, we present the results of both testsets.

5.1 Measures for Evaluation

5.1.1 Views Predicted

This metric shows how many views out of all views the recommendations predict. When we make a recommendation for a view event of a user, and the next item the user views is one of the three recommendations, then we assume that he would have clicked on this recommendation and this counts as a hit. The statistics outputs the number of times a hit occurs out of all views, in percent. The first view of the session is ignored because we can not make a recommendation for it. An example is shown below.

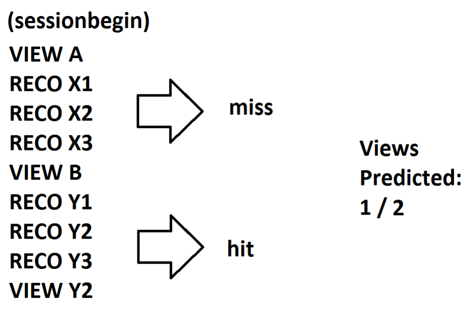


Figure 5.1: Example for predicted views statistic

5.1.2 Carts Predicted

This metric shows how many cart events out of all carts the recommendations predict. All recommendations of the session are stored. If the item of the cart event is contained in the set of recommendations that were made up until that cart, it counts as a hit. The idea is that if the user would have seen the recommendation, he would have taken a shortcut and go directly to the recommended item and then cart it. The metric outputs the number of carts predicted out of all carts, in percent. An example is shown below.

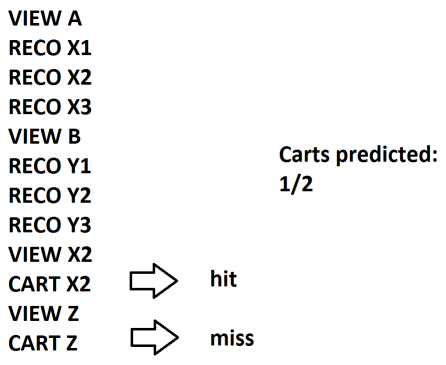


Figure 5.2: Example for predicted carts measure

5.1.3 Sales Predicted

This metric shows how many sale events out of all sales the recommendations predict. All recommendations of the session are stored. If the item of the sale event is contained in the set of recommendations that were made up until that sale, it counts as a hit. The idea is that if the user would have seen the recommendation, he would have taken a shortcut and go directly to the recommended item and then cart and buy it. The metric outputs the number of sales predicted out of all sales, in percent. An example is shown below.

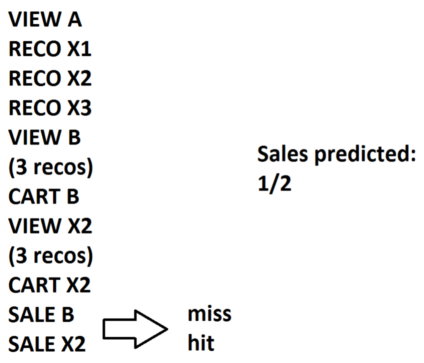


Figure 5.3: Example for predicted sales measure

5.2 Test Procedure

This section describes the methods that are tested, their possible combinations and the order of tests. Also further analysis is outlined.

Derived from the rules, there are a variety of different methods that are evaluated. The baseline are recommendations whose ranking is only based on the popularity count of the items.
Then, all possible combinations of simple rules are compared. Those are:
- View rule
- Cart rule
- Sale rule
- View+ Cart rule
- View + Sale rule

View + Cart + Sale and Cart + Sale are no useful combinations because the Sale rule is a subset of the Cart rule.

In addition to that, we compare the combinations above to our Conditional Function. In total, these are six methods to test.

Each of these six methods are tested with different history lengths. We test with a history length of zero, where only the currently looked at item is relevant, with a history length of one, where we also use the features of the item last looked at, and with a history length of two.

Each of the combinations above is tested with the algorithm outlined in section 4.6. We test them with one and two iterations of the algorithm, in other words with search depth 1 and 2. More iterations are not done due to time constraints and the poor performance of search depth 2 which will be discussed later.

Each method is tested with two classifiers, Naive Bayes and Hoeffding Trees, which are described in section 4.4.

In the next step, we determine the best method which predicts exactly one recommendation that will most likely be sold. We use the best method and combine it with the conditional rules to form six combinations of a Conditional Function as described in section 4.2.6. We determine the best combination, and then compare it to the best results of the simple rules.

In the next step, we try to further enhance the performance of rules by using random projection on the best Conditional Function and the best simple rule. We compare the results enhanced with random projection to the results without. We also use the methods with random projection and those without to analyse them regarding the impact of history length and search depth.

After that, we determine the two best methods in the three measurements ”Views Predicted”, ”Carts Predicted”, ”Sales Predicted”. This yields up to six different methods (or less, if the top two of the categories have overlap). We then use them to validate our results on the big set to see if the results stay the same or differ from the small set.

5.3 The Test Set

5.3.1 Small Set, Evaluated Locally

The small set consists of the first 0.5 percent of the whole log file. It contains roughly 2 million entries, about two thirds of those are recommendations from Zalando which are not used. The cleaned set consists of approximately 700.000 entries. The first 90 percent of the file are used for training, the last 10 percent are used for testing.

5.3.2 Big Set, Evaluated on Cluster

The big set consists of the whole Zalando Dataset. It contains roughly 400 million events, again two thirds of those are recommendations from Zalando which are not used. 90 percent of the events are used for training, the last 10 percent are used for testing.

5.4.1 Baseline, Maximum Possible Scores

The maximum possible scores are computed by returning all items as a recommendation, that are found in the first step of the algorithm (finding all items with a difference by one feature). On average, 120 items are found with one iteration, 794 with two. Following scores are achievable:

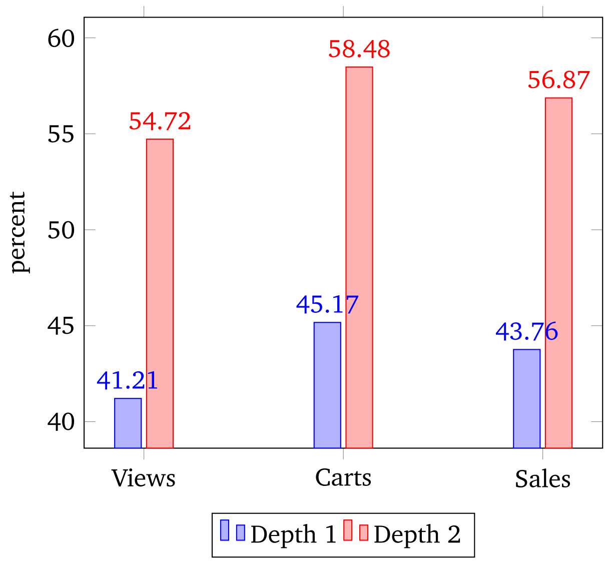


Figure 5.4: Maximum Possible Scores for Search Depth 1 and 2

Returning all items obviously is not a suitable baseline, because we must not recommend more than three items. Therefore the baseline is the top three items of the popularity count of the items. Every view, cart and sale event is used to compute the score. Example: Item A is viewed 10 times, carted 1 time and sold 0 times. Item B is viewed 7 times, carted 4 times and sold 1 time. Item A then has a score of 10+1+0=11, item B has a score of 7+4+1=12. The higher the score, in other words the more popular an item is, the higher it is ranked. We chose this baseline because it is independent of any other parameters like history length. Essentially, we just recommend items that are popular in general. Following scores are achieved:

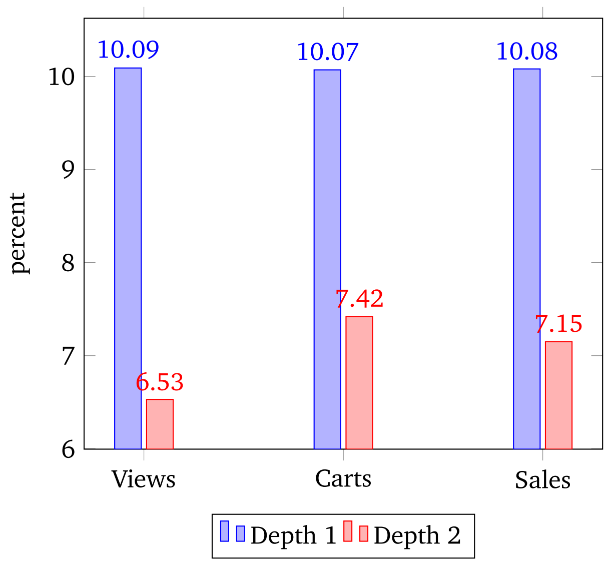


Figure 5.5: Achievable scores in the measures ”Views Predicted”, ”Carts Predicted” and ”Sales Predicted” for the baseline

5.4.2 Simple Rules

We now compare the simple rules, namely View, Cart, Sale, View+Cart, View+Sale. We compare them regarding their performance in the categories ”Views Predicted”, ”Carts Predicted” and ”Sales Predicted”. For each history length, we do a comparison with both search depth 1 and 2. We will determine a winner for each history length, and then determine an overall winner, which is used for later evaluations and an evaluation on the big test set.

Search Depth 1:

Naive Bayes is inferior to Hoeffding Trees. Most direct comparisons of a method regarding the two classifiers, Hoeffding Trees do better. However, this is not the case for predicting carts. Naive Bayes does better here when looking at the Cart and View+Cart rule.

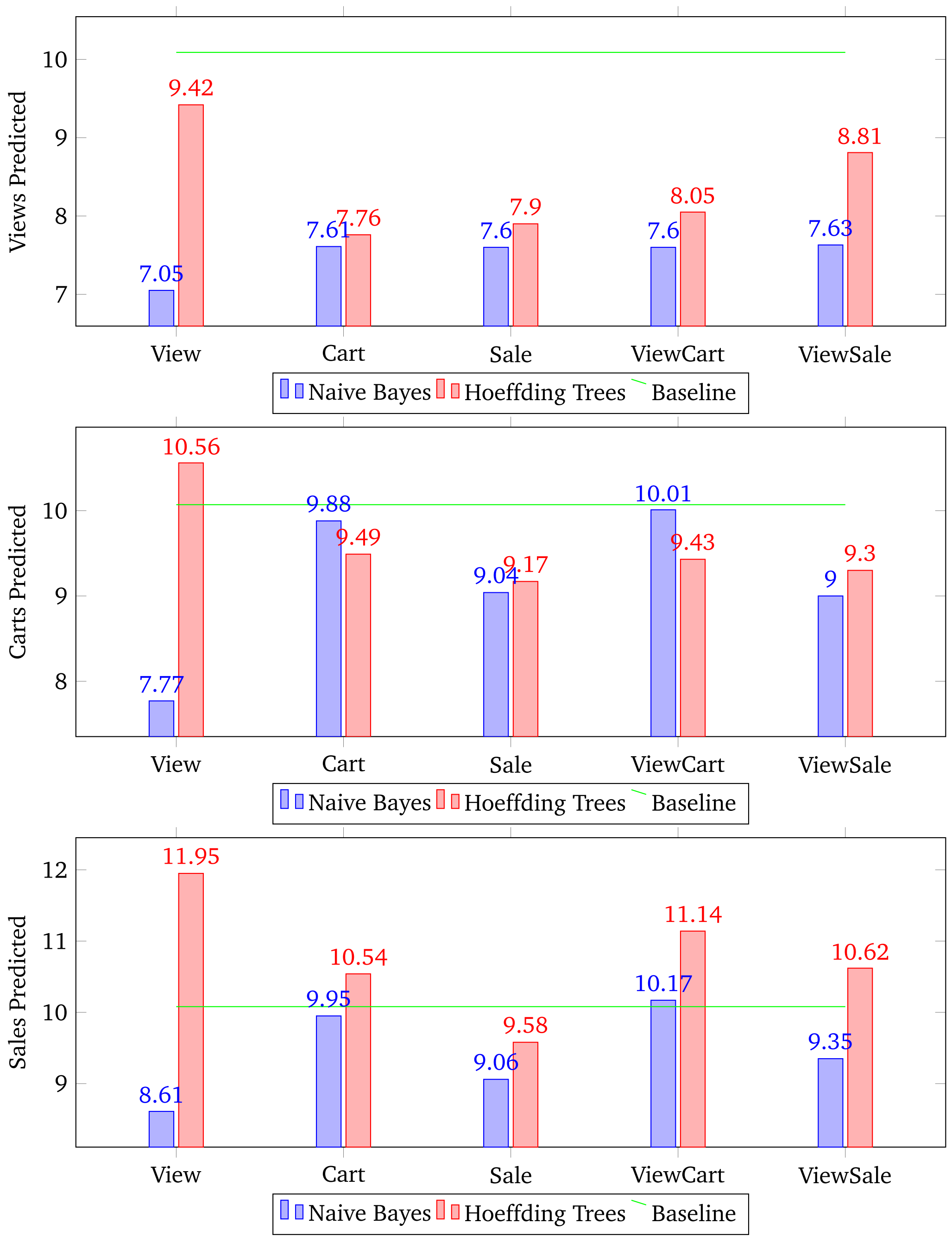


Figure 5.6: Scores of the simple rules with the parameters history length 0, search depth 1

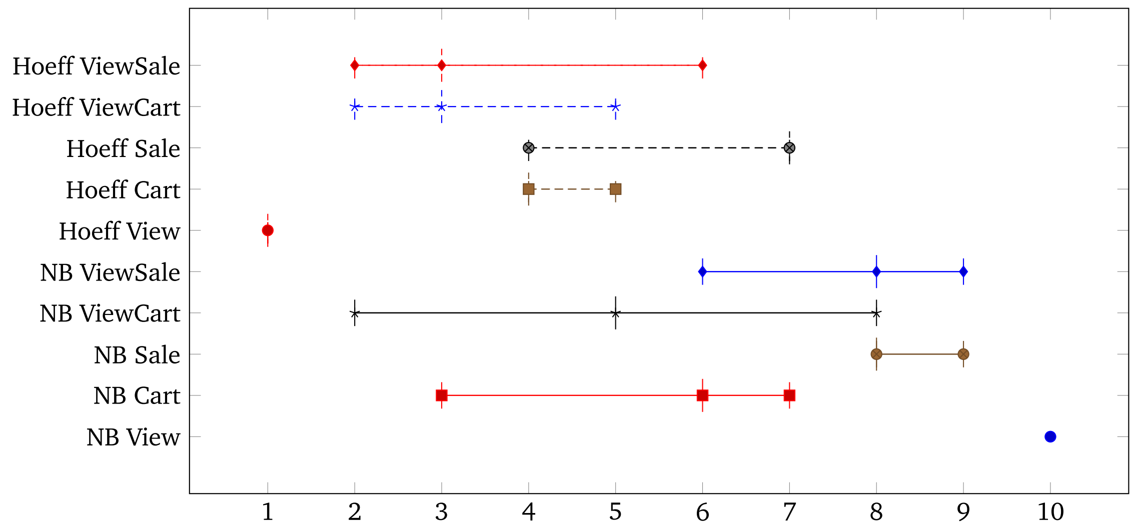


Figure 5.7: Ranking of simple rules with the parameters history length 0, search depth 1. The rank in every measurement is used to compute the median.

Looking at Naive Bayes, the View+Cart rule outperforms every other method. View+Cart does best regarding carts and sales predicted, which is to be expected looking at the way the rule works. It is interesting that Cart outperforms the View rule as well as the View+Cart rule - although only by a very slight margin - in the statistic ”Views Predicted”. Combining the View and Sale rules does not bring any performance improvements.

Looking at Hoeffding Trees, the View rule does best. It outperforms all other methods in views predicted, which is to be expected looking at the way the rule works. However, it also outperforms all other rule combinations with regards to carts/sales predicted. This might be due to the higher amount of training examples the View rule produces. The classifier therefore can learn a better model. A combination of rules (View+Cart and View+Sale) does not lead to performance improvements.

Summarising, there is one clear winner for each classifier: The View+Cart rule for Naive Bayes and the View rule for Hoeffding Trees. Except for two times, Naive Bayes is always inferior to Hoeffding Trees. Overall, Hoeffding Trees with View rule is the winner.

Search Depth 2:

Doing one more iteration of the algorithm results in a significant drop in all measurements.

Again, Naive Bayes is inferior to Hoeffding Trees. There is only one case where Naive Bayes surpasses Hoeffding Trees: the Sale rule does slightly better at predicting carts.

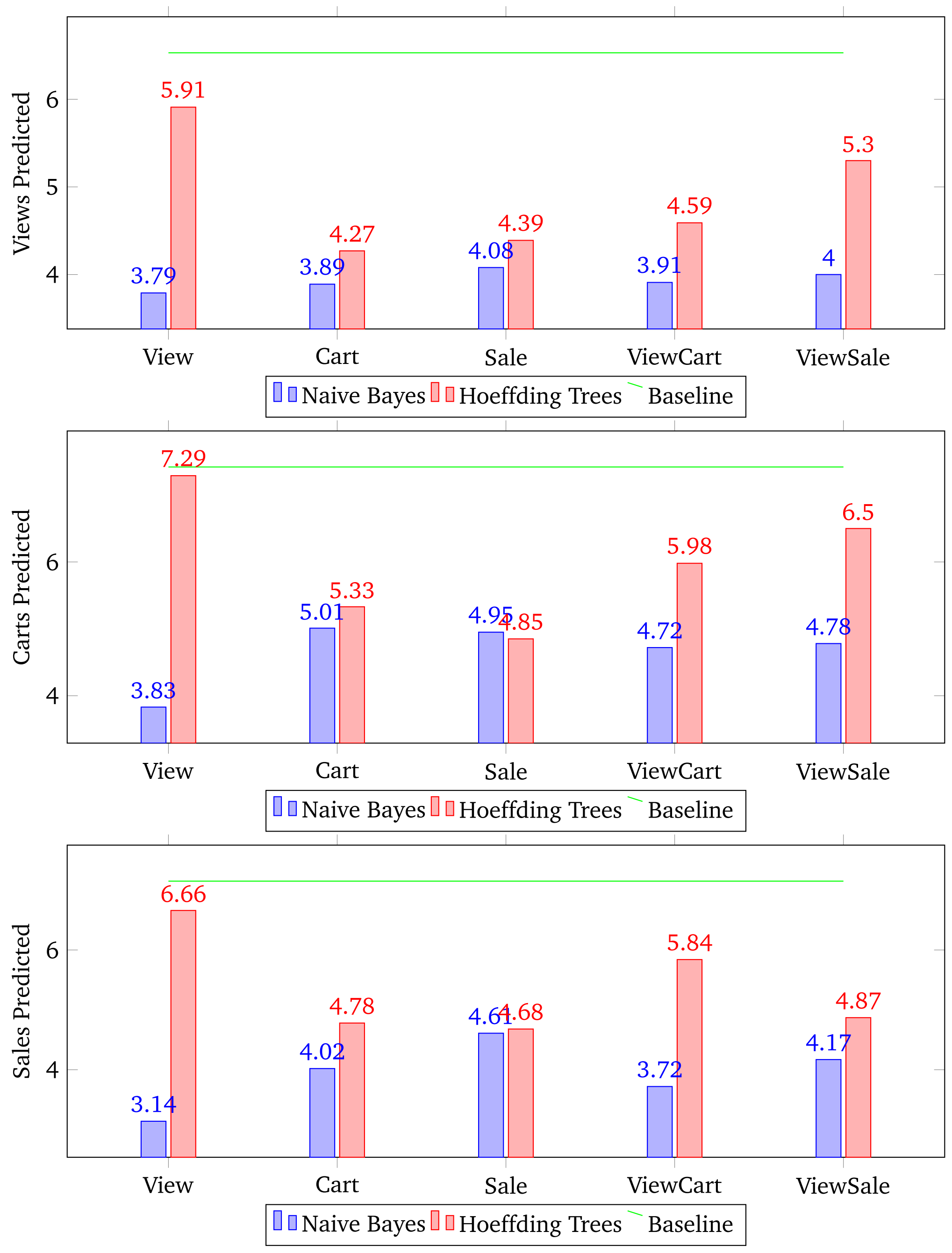


Figure 5.8: Scores of the simple rules with the parameters history length 0, search depth 2

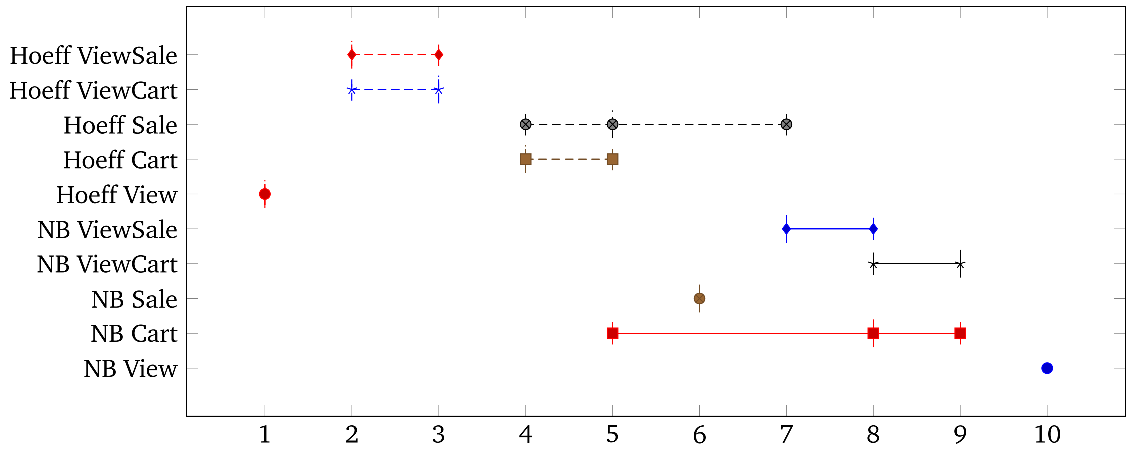


Figure 5.9: Ranking of simple rules with the parameters history length 0, search depth 2. The rank in every measurement is used to compute the median.

Looking at Naive Bayes, there are two winners. The Cart rule does best regarding "Carts predicted", while the Sale rule does best regarding "Sales predicted". The Sale rule also does best at predicting views - one would expect the View rule to be the winner here. The View rule however is last in all categories, which might be due to the high number of conflicting training examples the View rule produces, to which Naive Bayes is sensitive.

Looking at Hoeffding Trees, the View rule is again the winner, this time with a clear lead in all categories. In comparison with "View Rule Search Depth 1", a significant drop in "Views Predicted" is noticeable. The drop in "Carts Predicted" and "Sales Predicted" is not as substantial, which might be due to the overall lower number of carts and sales.

Summarising, there are two winners for Naive Bayes. The Cart/Sale rules do best in their respective categories, with the Sale rule also being best at predicting views. They all are inferior to the best Hoeffding Trees results though, in which the View rule outperforms all other methods. Once again, combinations of rules do not bring any performance improvements for Hoeffding Trees. All results are inferior to the baseline.

Summary

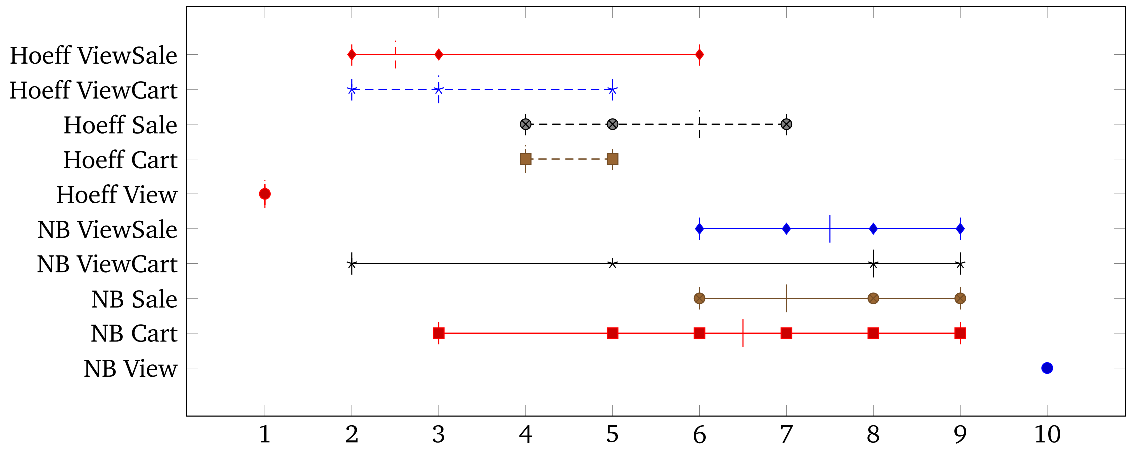


Figure 5.10: Ranking of simple rules with history length 0, regardless of search depth. The rank in every measurement is used to compute the median.

The View rule trained with Hoeffding Trees is the clear winner with rank 1 in every statistic, both with depth 1 and 2. Interestingly, the View rule trained with Naive Bayes does worst with rank 10 in every statistic. All median results of Hoeffding Trees are better than Naive Bayes, which shows that Naive Bayes generally performs worse here.

Search Depth 1:

Again, Naive Bayes is inferior to Hoeffding Trees. Except for one case, in each direct comparison of a method regarding the two classifiers, Hoeffding Trees do better. The Cart rule trained with Naive Bayes does slightly better at predicting carts than its Hoeffding Trees counterpart.

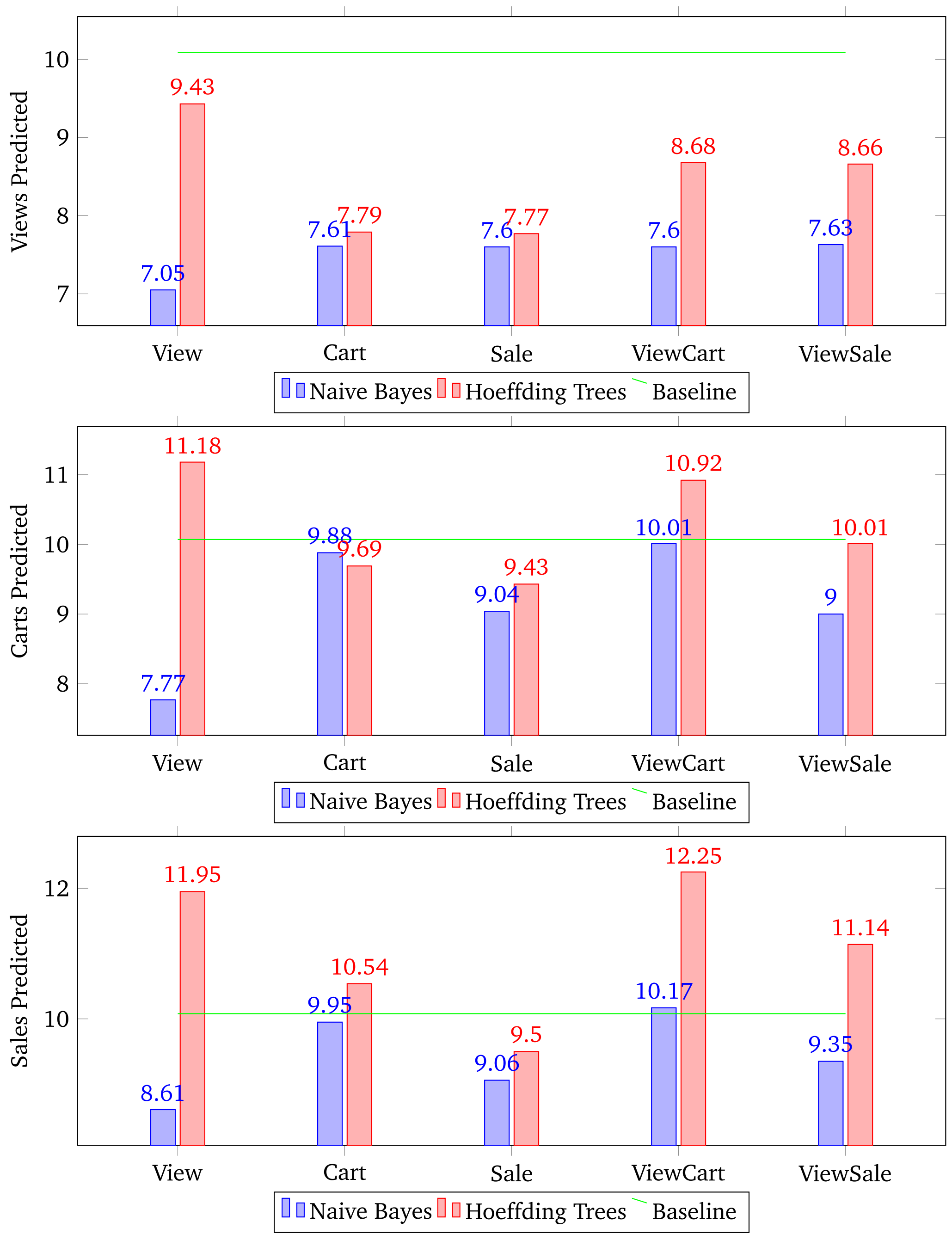


Figure 5.11: Scores of the simple rules with the parameters history length 1, search depth 1

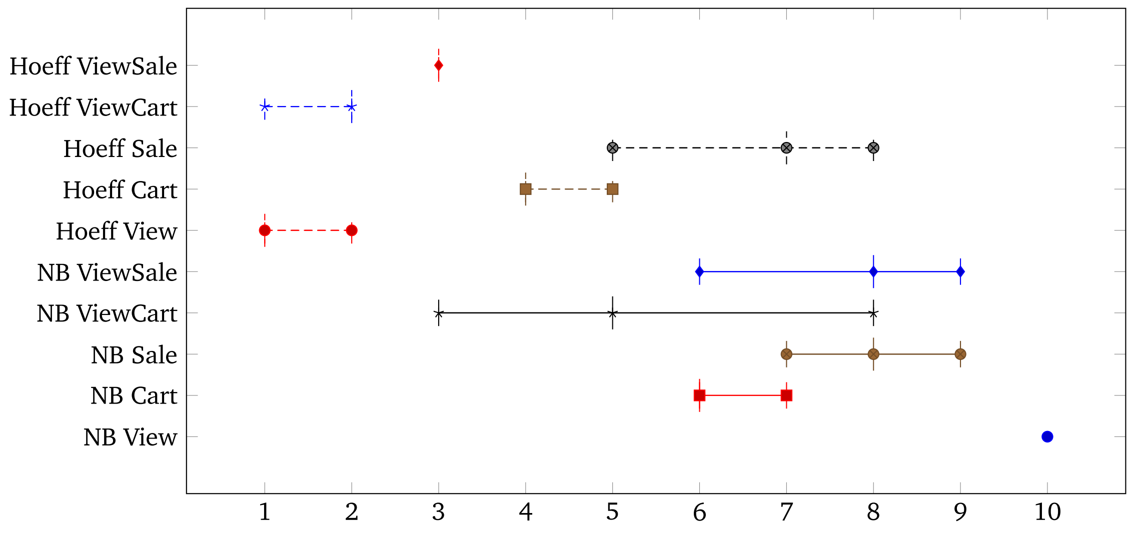


Figure 5.12: Ranking of simple rules with the parameters history length 1, search depth 1. The rank in every measurement is used to compute the median.

Looking at Naive Bayes, the results are exactly equal to history length 0. An increase of the history length does not bring any improvements. The View+Cart rule does best at predicting carts and sales, and is almost even with the View+Sale rule at predicting views.

Looking at Hoeffding Trees, the View rule is once again does best, but not with a clear lead like before. It outperforms all other methods except the View+Cart rule, which is slightly better in the category "Sales Predicted". This may be due to the fact that cart events are a subset of sale events, and the learner can utilise the additional information with a higher history length to predict sales better. This means that now a combination of rules leads to a performance improvement.

Summarising, there is one winner for each classifier: The View+Cart rule for Naive Bayes and the View rule for Hoeffding Trees, with the results for Naive Bayes being exactly the same as the results of history length 0.

Search Depth 2:

Doing one more iteration of the algorithm results in a significant drop in all measurements.

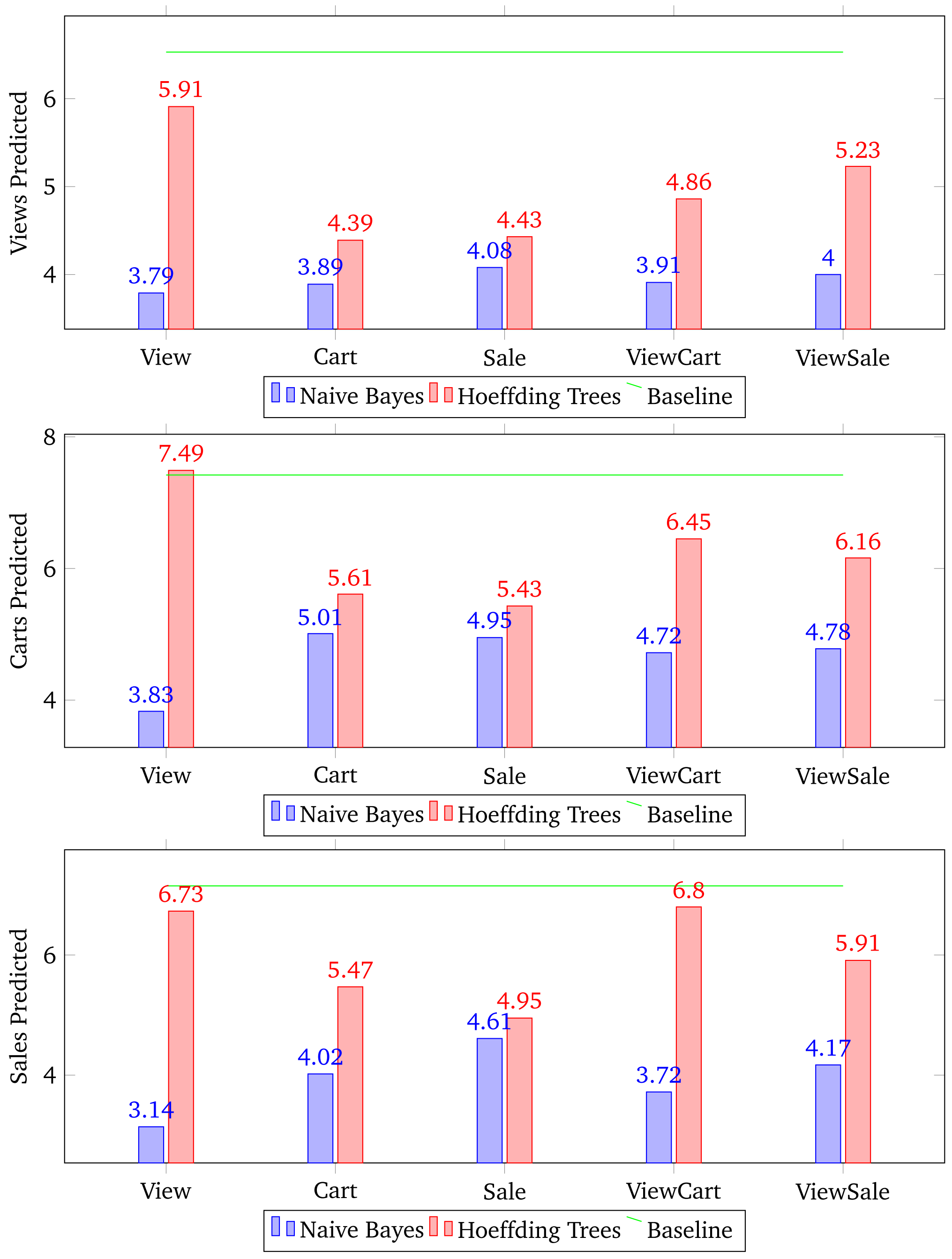


Figure 5.13: Scores of the simple rules with the parameters history length 1, search depth 2

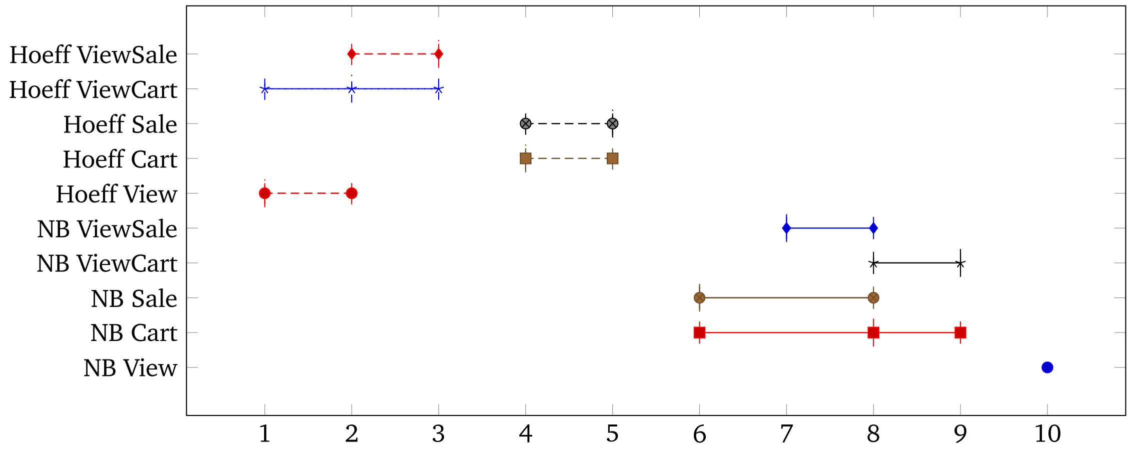


Figure 5.14: Ranking of simple rules with the parameters history length 1, search depth 2. The rank in every measurement is used to compute the median.

Looking at Naive Bayes, there are two winners. Similar to the results of Depth 1, the history length does not bring any performance improvements, the results are exactly the same as with history length 0. The Cart rule does best regarding ”Carts Predicted”, while the Sale rule does best regarding ”Sales Predicted”.

Looking at Hoeffding Trees, the View rule is again the winner, this time with a clear lead in all categories except the predicted sales, where View+Cart does best. Ranking-wise, these results are equal to search depth 1. In comparison with ”View Rule Search Depth 1”, a significant drop in all categories is noticeable.

Summarising, there are two winners for Naive Bayes. The Cart/Sale rules do best in their respective categories. They all are inferior to the Hoeffding Trees results though, in which the View rule outperforms most other methods.

Summary

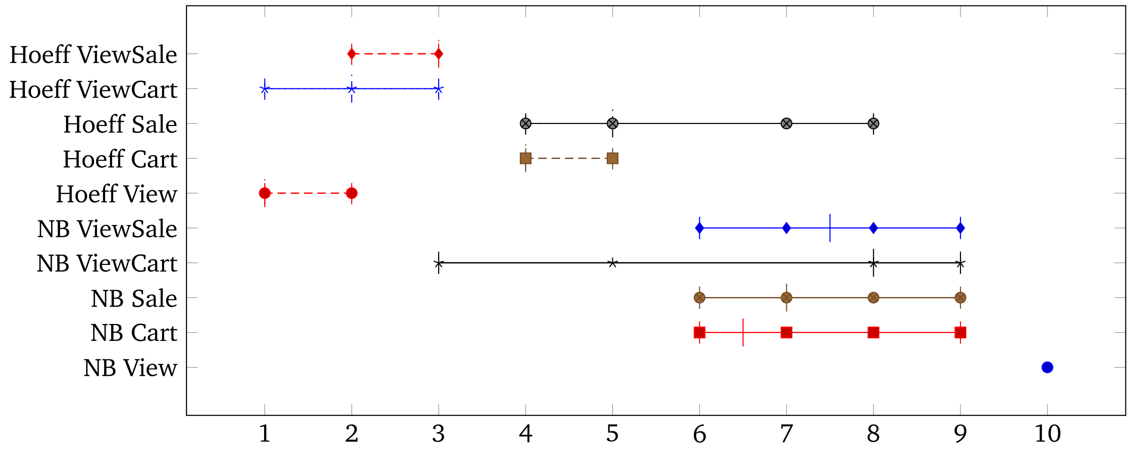


Figure 5.15: Ranking of simple rules with history length 1, regardless of search depth. The rank in every measurement is used to compute the median.

The View rule trained with Hoeffding Trees again is the winner. It is ahead in ”Views/Carts Predicted” , only beaten by the View+Cart rule in the category ”Sales Predicted”. All median results of Hoeffding are better than Naive Bayes. For the first time, a combination of rules leads to performance improvements for Hoeffding Trees. Apart from that, the results are almost equal to those of history length 0, for Naive Bayes the results are even exactly the same.

Search Depth 1:

Again, Naive Bayes is inferior to Hoeffding Trees. Except for one case, in each direct comparison of a method regarding the two classifiers, Hoeffding Trees do better. The Cart rule trained with Naive Bayes does slightly better at predicting carts than its Hoeffding Trees counterpart.

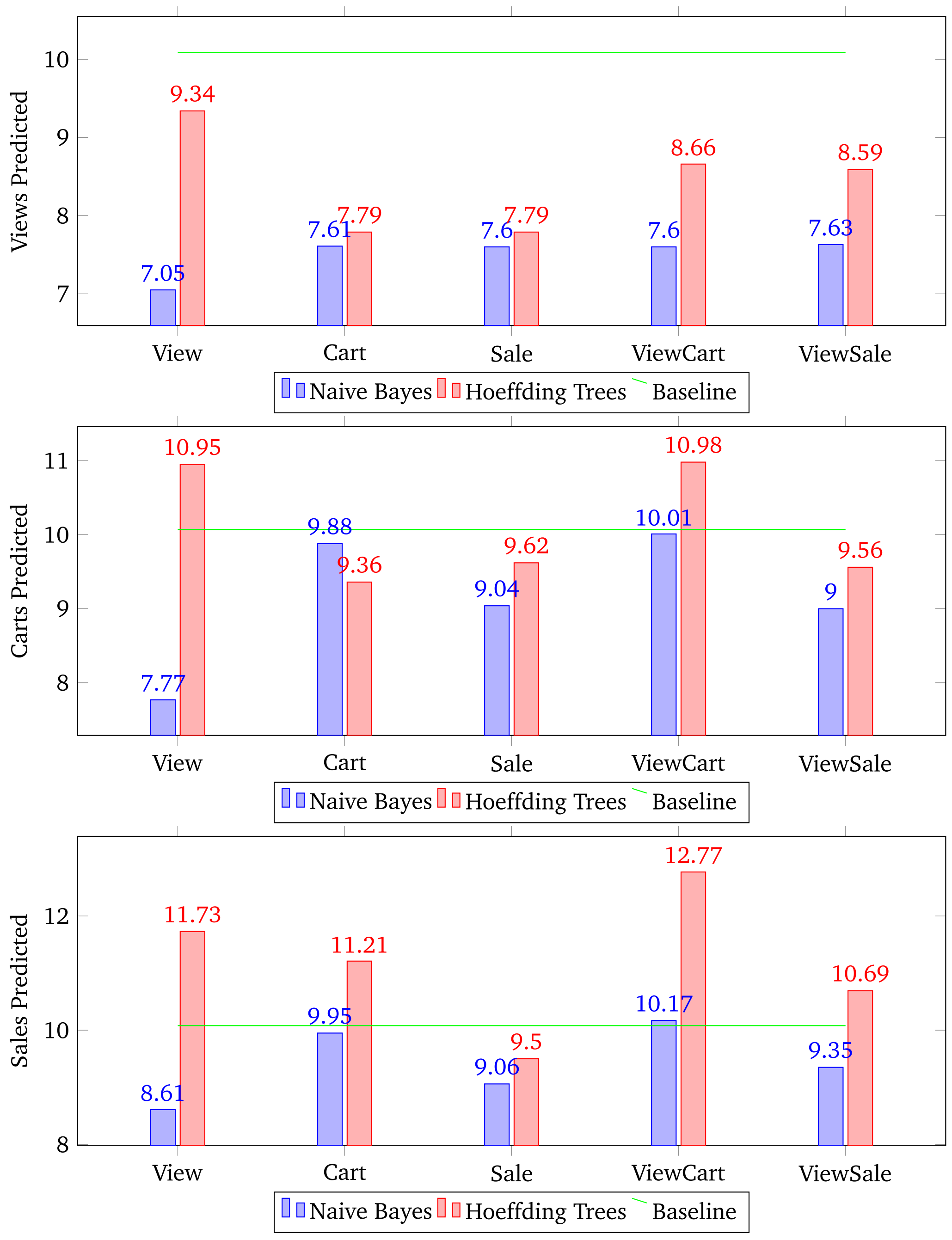


Figure 5.16: Scores of the simple rules with the parameters history length 2, search depth 1

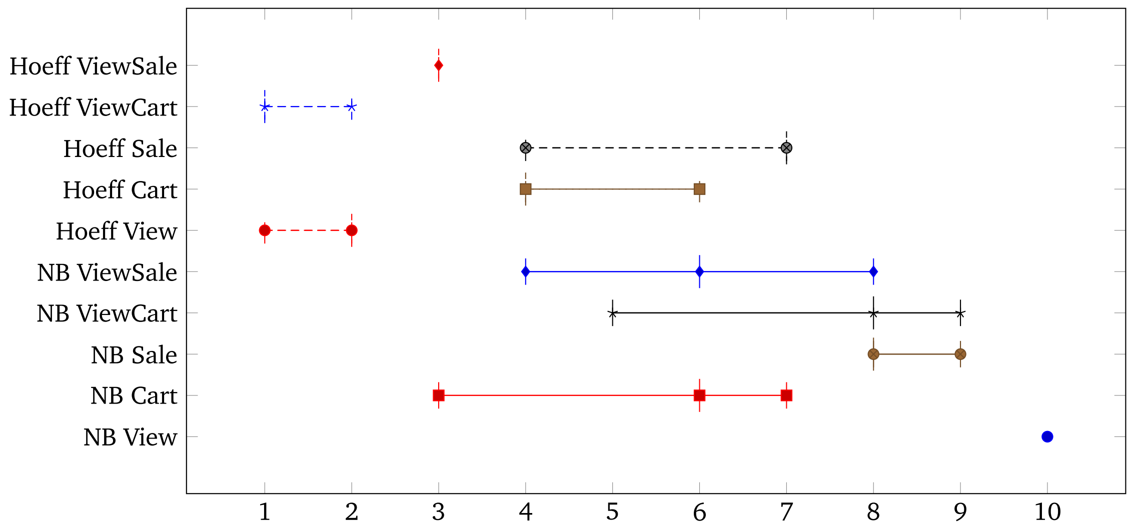


Figure 5.17: Ranking of simple rules with the parameters history length 2, search depth 1. The rank in every measurement is used to compute the median.

Looking at Naive Bayes, the results are exactly equal to history length 0 and 1. An increase of the history length does not bring any improvements. The View+Cart rule does best at predicting carts and sales, and is almost even with the View+Sale rule at predicting views.

Looking at Hoeffding Trees, the View+Cart rule does best, with the View rule coming in close second. View is much better at predicting views, and only slightly worse at predicting carts. View+Cart also does best at predicting sales: Again, a combination of rules leads to a performance improvement. Compared to history length 1, there is a drop in performance for Hoeffding Trees. This is the first time, a rule other than View is leading.

Summarising, there are two winners for each classifier: The View+Cart rule for Naive Bayes and the View+Cart rule for Hoeffding Trees. A combination of rules leads to performance improvements for Hoeffding Trees. Compared to history length 1, there is a drop in performance for Hoeffding Trees.

Search Depth 2:

Again, doing one more iteration of the algorithm results in a significant drop in all categories.

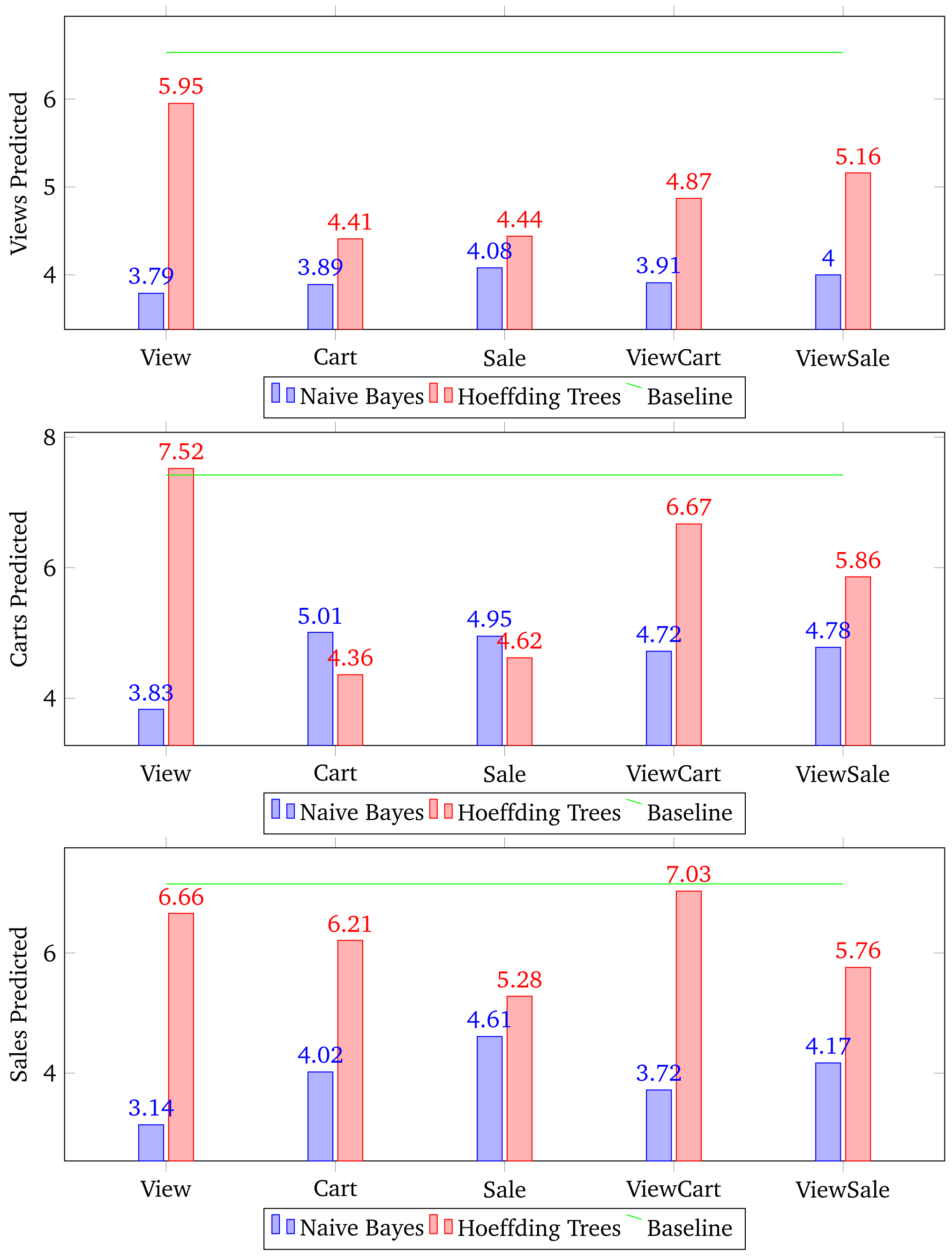


Figure 5.18: Scores of the simple rules with the parameters history length 2, search depth 2

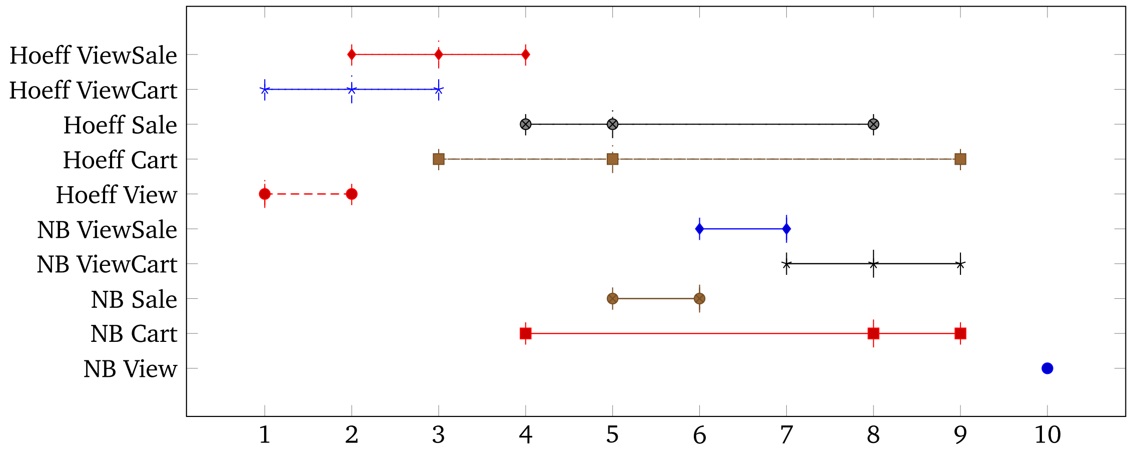


Figure 5.19: Ranking of simple rules with the parameters history length 2, search depth 2. The rank in every measurement is used to compute the median.

Looking at Naive Bayes, there is no clear winner. Again, the history length does not bring any performance improvements, the results are exactly the same like with history length 0 and 1. The Cart rule does best regarding ”Carts Predicted”, while the Sale rule does best regarding ”Sales Predicted”.

Looking at Hoeffding Trees, the View rule is again the winner, this time with a clear lead in all categories except the predicted Sales, where View+Cart does best. In comparison with ”View Rule Search Depth 1”, a significant drop in all categories is noticeable. Compared to history length 1, the results are worse.

Summarising, there are two winners for Naive Bayes. The Cart/Sale rule do best in different categories. They all are inferior to the Hoeffding Trees results though, in which the View rule outperforms most other methods. A combination of rules leads to performance improvements for Hoeffding Trees. Compared to history length 1, there is a drop in performance for Hoeffding Trees.

Summary

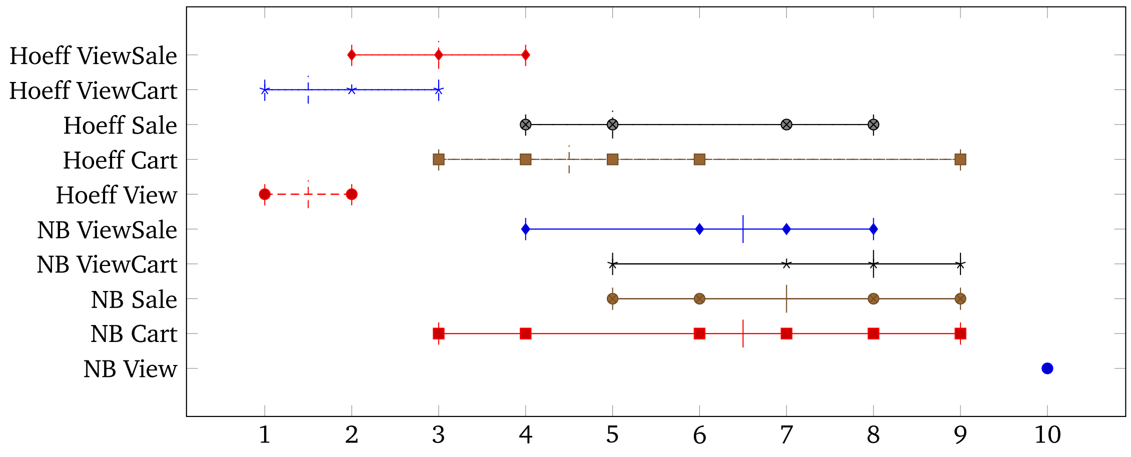


Figure 5.20: Ranking of simple rules with history length 2, regardless of search depth. The rank in every measurement is used to compute the median.

The View rule trained with Hoeffding Trees again is the winner, with the View+Cart rule trained with Hoeffding Trees in close second. The View+Cart rule does better at predicting sales, with two wins in that category. The View rule does better in the category ”Views Predicted”. Both tie in the category ”Carts Predicted”. All median results of Hoeffding Trees are better than Naive Bayes. Hoeffding Trees experience a slight decrease in performance compared to history length 1.

Simple Rules Summary

Rules trained with Hoeffding Trees clearly outperform rules trained with Naive Bayes. In direct comparison, the median result of rules trained with Hoeffding Trees is always better than Naive Bayes. Because of that, we will ignore Naive Bayes in further evaluations. An increase of the search depth leads to a drop in performance in all categories. We will analyse this in more detail in section 5.4.6.

Overall, the View rule trained with Hoeffding Trees performs best and outperforms most other methods. It is only beaten by View+Cart trained with Hoeffding Trees in the category "Sales Predicted".

5.4.3 Conditional Function

Conditional Functions in Comparison

We first determine, which of the simple rules and the baseline does best at predicting sales, when making only one recommendation. The best method is chosen for the Conditional Function. The reasoning is as described in section 4.2.6 about Conditional Functions: We first want to recommend an item that will most likely be sold. We will only look at the results trained with Hoeffding Trees and the baseline, Naive Bayes is ignored because of the bad results. We show the results with history length 0.

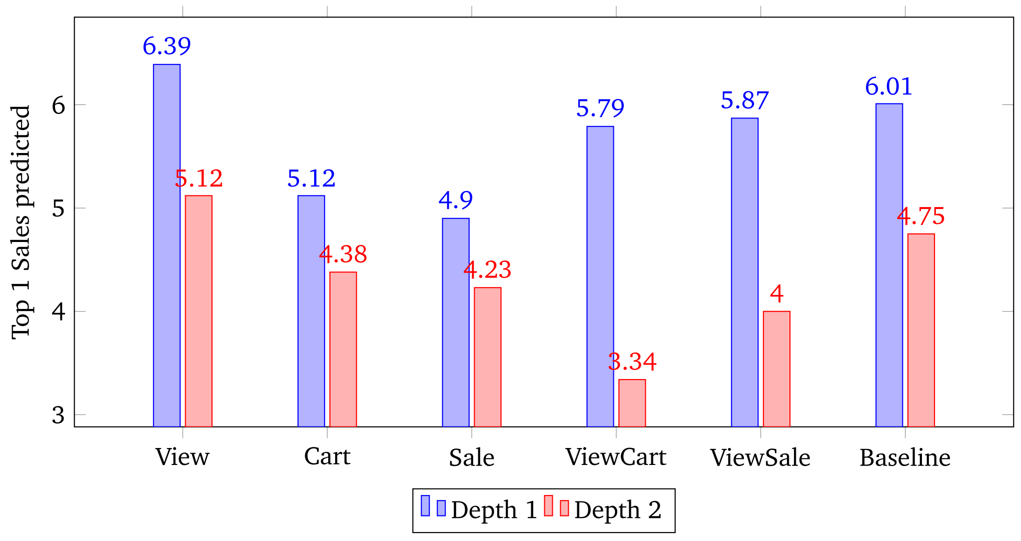


Figure 5.21: Performance of simple rules when predicting sales only making one recommendation

Looking at the results, we see interesting results compared to the results seen earlier. The View rule is best at recommending one item, both with depth 1 and 2. This is interesting because the View+Cart rule is best at recommending 3 items that will likely be sold later. Also the View rule doing 3 recommendations with search depth 2 is worse than the baseline. But looking only at the first recommendation made, the View rule does better. It seems, that the ordering at the very top of the ranking is done better by the View rule.

Deriving from the results, we will use the View rule to make our first recommendation. The second and third rule are the Conditional Cart / Conditional Cart No Sale, and Conditional View / Conditional View No Cart Sale / Conditional View No Cart Sale Subset rules, which are discussed in section 4.2.6. This yields 6 combinations, which are abbreviated as followed:

V-C-V	View / Conditional Cart / Conditional View
V-C-VNC	View / Conditional Cart / Conditional View No Sale
V-C-VNCS	View / Conditional Cart / Conditional View No Sale Subset
V-CNS-V	View / Conditional Cart No Sale / Conditional View
V-CNS-VNC	View / Conditional Cart No Sale / Conditional View No Cart Sale
V-CNS-VNCS	View / Conditional Cart No Sale / Conditional View No Cart Sale Subset

Table 5.1: List of abbreviations of all Conditional Functions

The Conditional Functions are compared using history length 0. We will not compare them with history length 1 and 2 due to limited time. We compare them regarding the Views / Carts / Sales predicted.

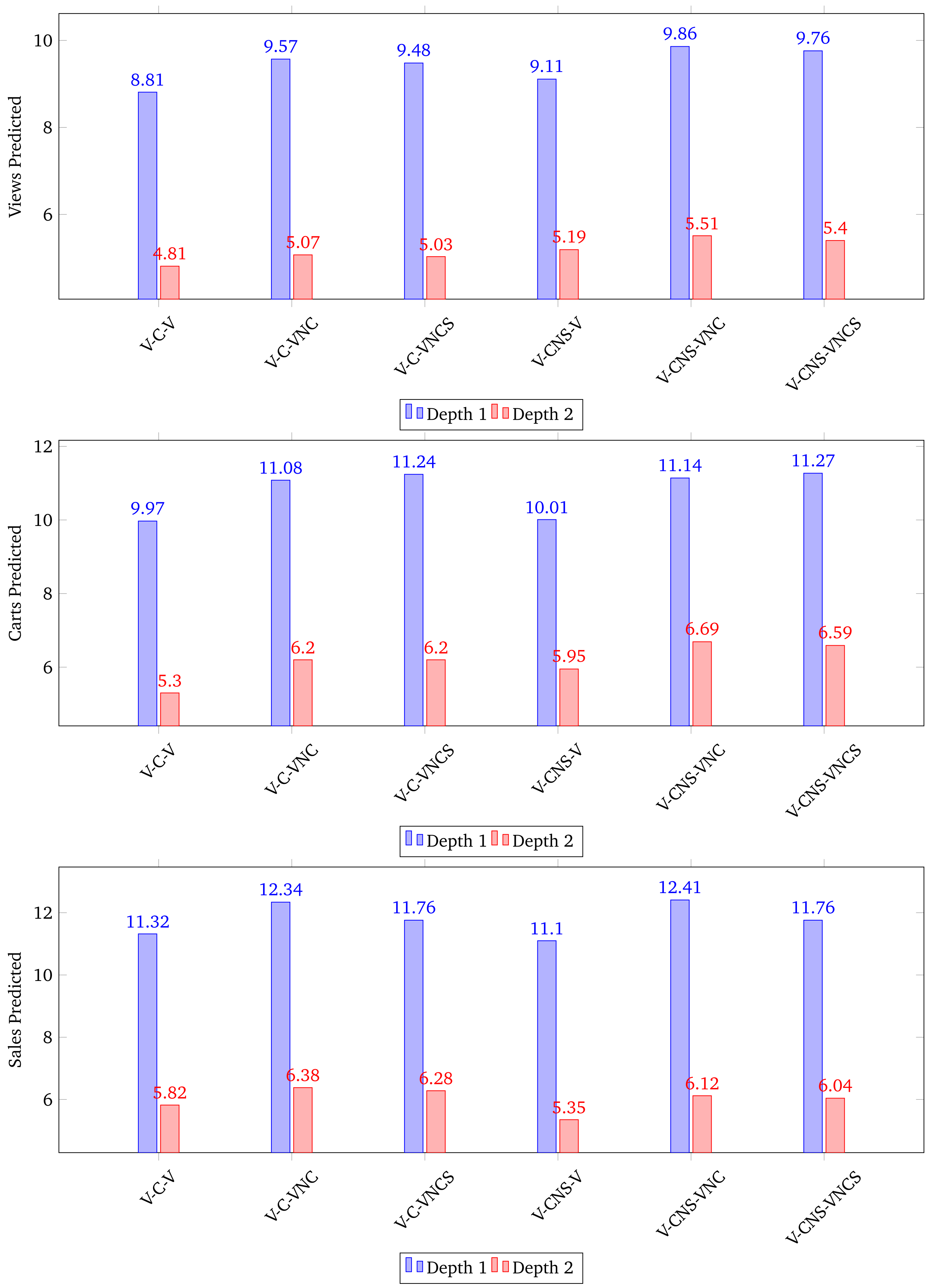


Figure 5.22: Conditional functions in comparison and their scores in all measurements

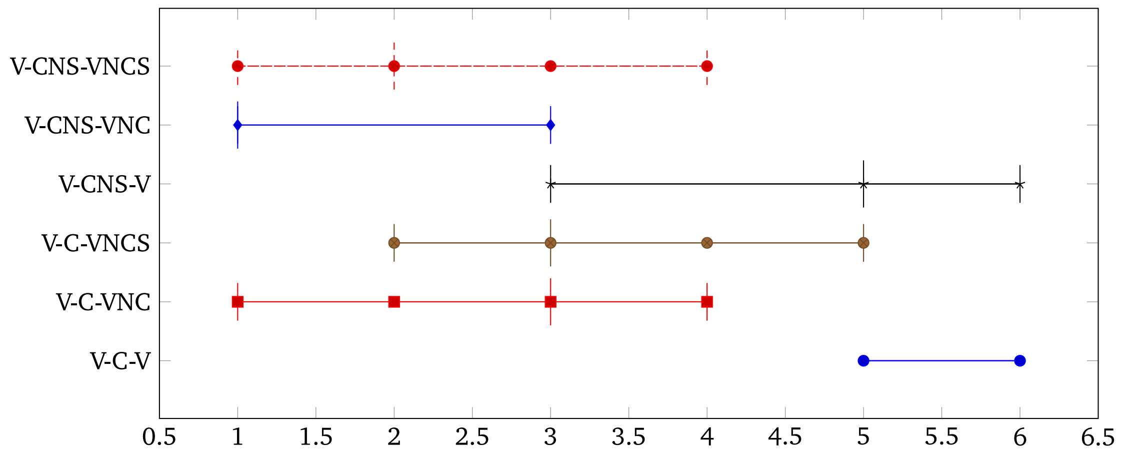


Figure 5.23: Ranking of conditional functions

Regarding search depth 1, the Conditional Function ”View / Conditional Cart No Sale / Conditional View No Cart Sale” is the winner, performing best in the categories ”Views Predicted” and ”Sales Predicted”. It is third in the category ”Carts Predicted”, surpassed by V-CNS-VNCS, which is second overall, and V-C-VNCS. Similar to our findings for the simple rules, depth 2 always does worse than depth 1 in direct comparison, but the decrease is more substantial. A reason for this is that every Conditional Function consists of three rules, which each have to narrow an average of 794 items down to 1 recommendation. It seems that this yields too many errors in comparison to search depth 1, where on average only 120 items are taken into account. Because of these results, a good rank for search depth 1 is more important than a good rank for search depth 2. Looking at depth 2, the Conditional Function ”View / Conditional Cart No Sale / Conditional View No Cart Sale” again performs best with first in ”Views Predicted”, and ”Carts Predicted”, and third ”Sales Predicted”. Consequently, overall the combination ”View / Conditional Cart No Sale / Conditional View No Cart Sale” performs best. It wins in 4 out of 6 categories, 2 of which are the more important ones with search depth 1. Therefore, it is chosen as the Conditional Function.

Conditional Function in Comparison to best Simple Rule Results

We now compare the performance of the Conditional Function with the best results determined in the last section.

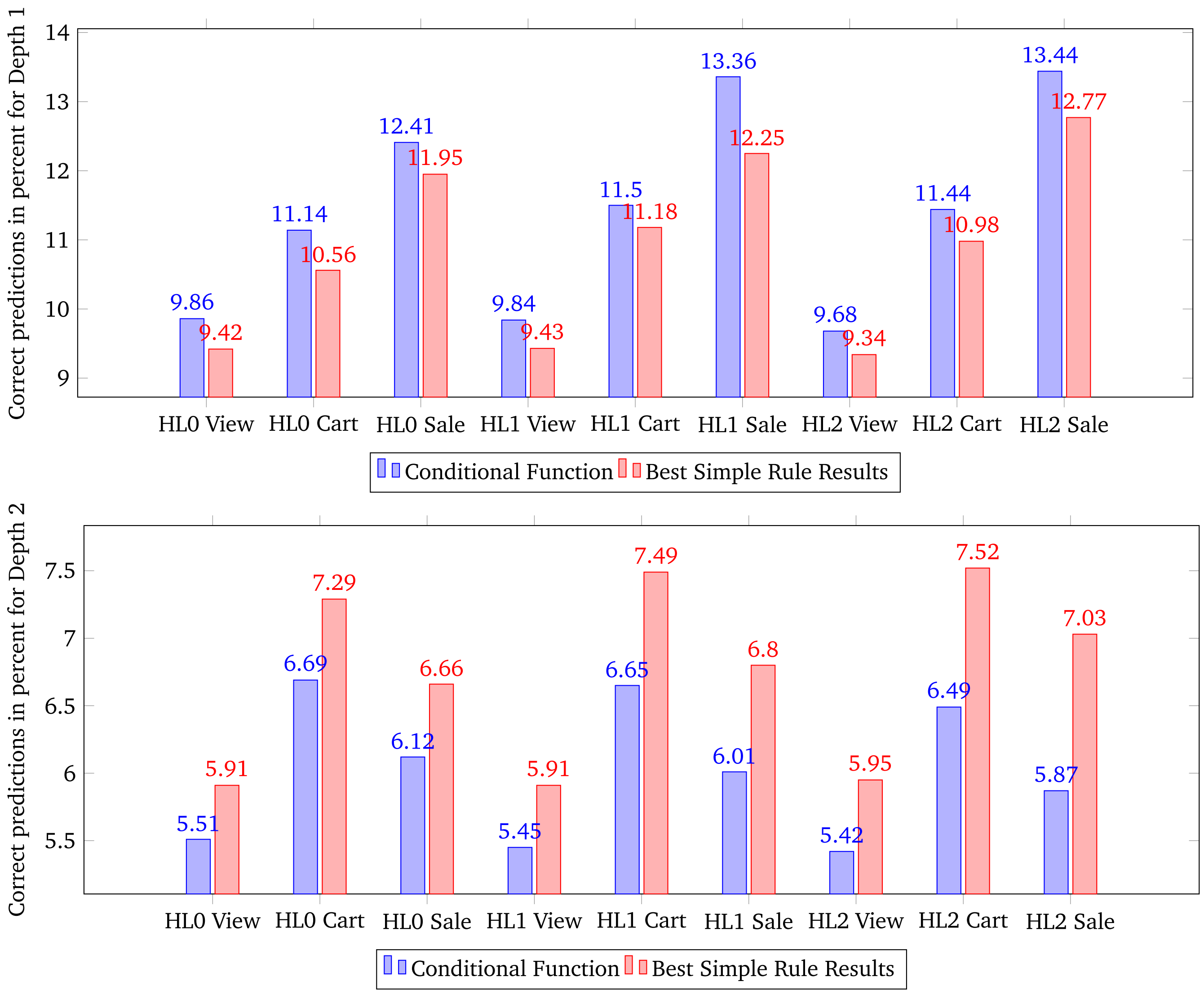


Figure 5.24: Conditional Function in comparison to best results of simple rules

There is one clear winner for each search depth. At search depth 1, the Conditonal Function outperforms the simple rules in every category. The Conditional Function's predicted sales profits from the history length. It seems that the Conditional Function can use the browsing history very well to predict what will be bought later, while the simple rules cannot profit as much from it. Both the simple rules and the Conditional Function have their overall strongest performance with history length 1.

Looking at search depth 2, the results are the opposite to search depth 1. The simple rules outperform the Conditional Function in every category. As discussed earlier, it seems that it is too hard to narrow down on average 794 items down to 1 recommendation, which the Conditional Function has to do three times to make its three recommendation. Because of this, there is a huge drop in performance in all categories for the Conditional Function. Simple rules on the other hand have to narrow down these on average 795 items down to only 3 recommendations one time, hence the drop in performance for simple rules is not as big in comparison to the Conditional Function.

Overall, the Conditional Function outperforms the simple rules. The Conditional Function has the highest results in the categories ”Views Predicted”, ”Carts Predicted” and ”Sales Predicted”.

5.4.4 Impact of Random Projection

Among the rules, the View rule trained with Hoeffding Trees yields the best results. Among Conditional Functions, ”View / Cart No Sale / View No Cart Sale” yields the best results. We try to further enhance their performance by doing a random projection and training the results with Hoeffding Trees. We do a projection to 300 dimensions and test the results with the history length 0 to 2 with both iteration depth 1 and 2.

View Rule

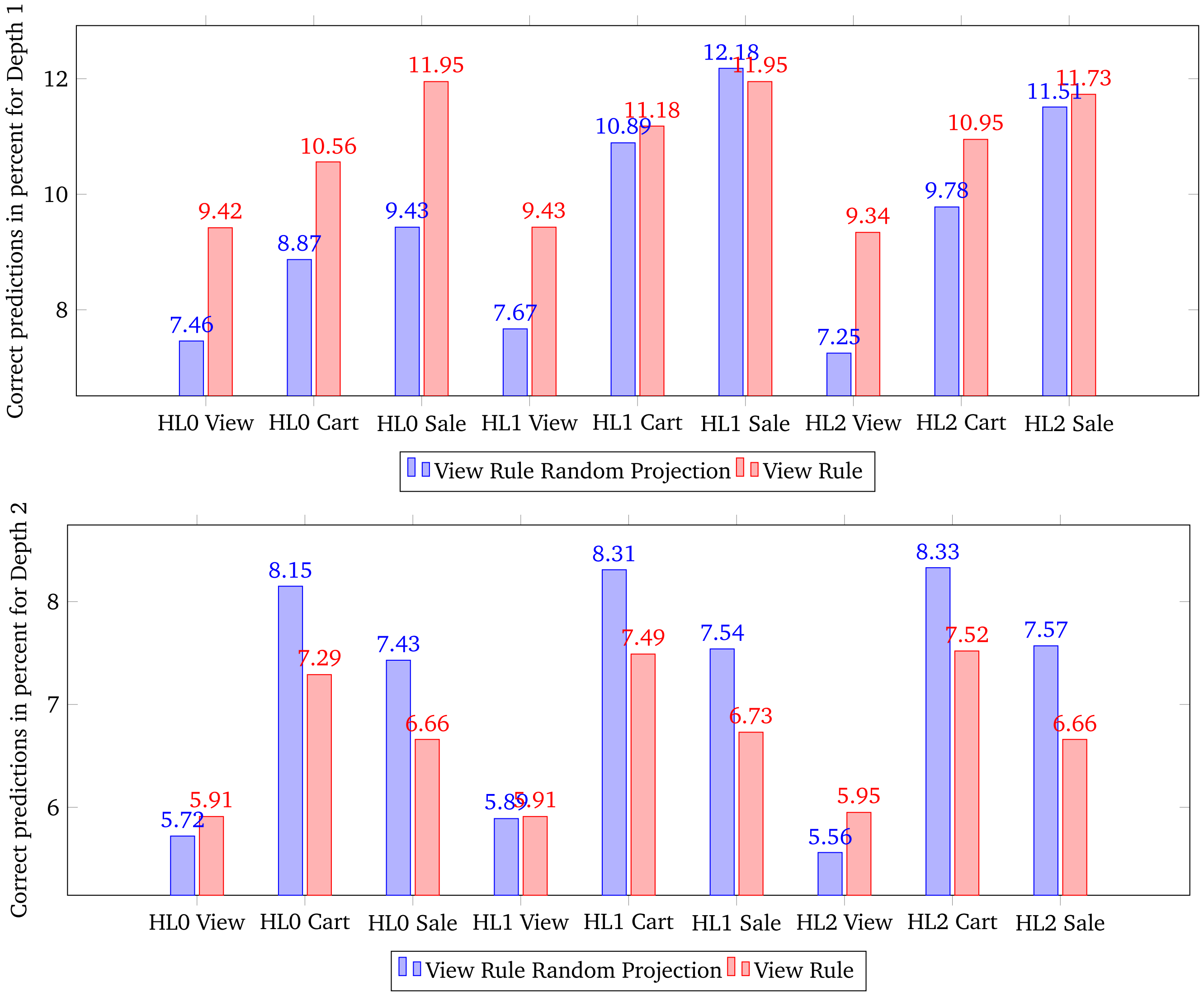
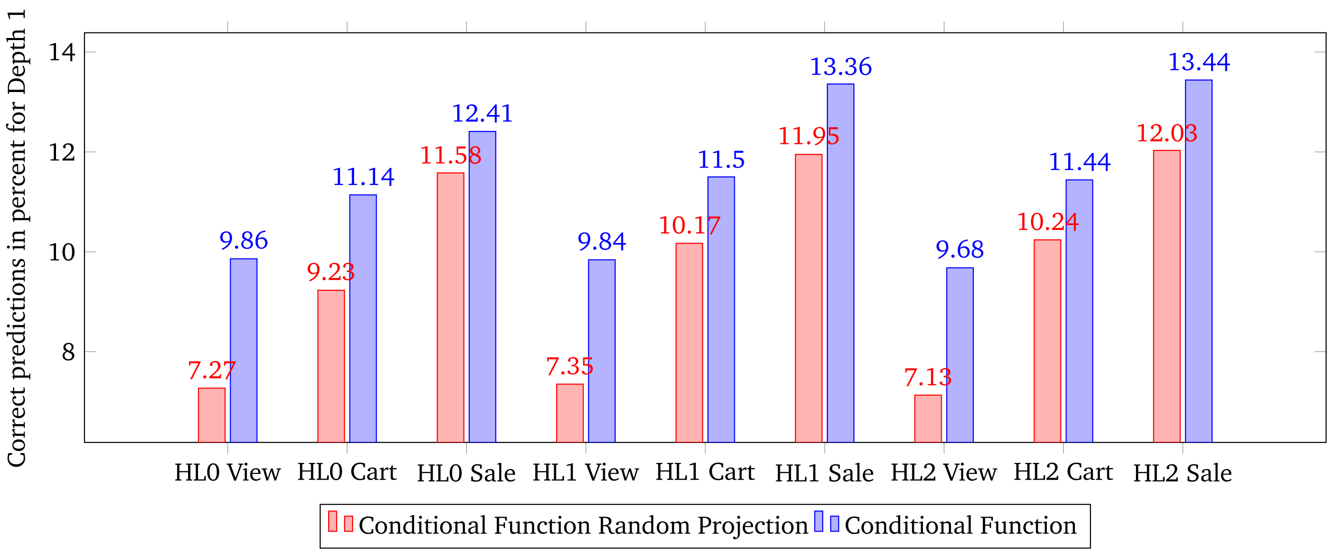


Figure 5.25: View rule comparison: Rule with random projection to rule without

The View rule outperforms the random projection in every category in search depth 1, except ”Predicted Sales, History length 1”. ”Predicted Sales” is also the category in which the random projection is closest to the View rule generally. Random projection is heavily influenced by the history length, with the best results with history length 1. Random projection is significantly worse at predicting views.

Looking at search depth 2, we see an increase in performance by the random projection compared to the View rule. The correct predictions for carts and sales of the random projection now surpass those of the View rule and increases with a longer history length. It seems that the history length is especially useful for predicting carts and sales, but not so much for predicting views. While ”Carts/Sales predicted” increase steadily, ”Views Predicted” only increases with history length 1 and then decreases again. Random projection seems to exploit the history length way better than the view Rule.

Conditional Function



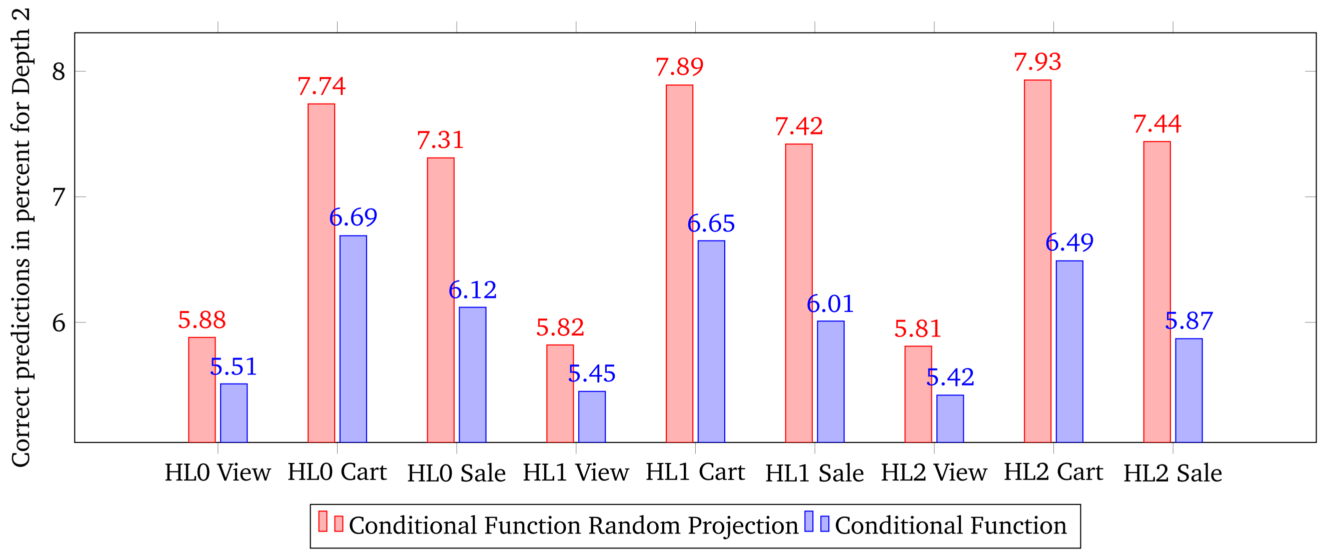


Figure 5.26: Conditional function: Comparison of rule with random projection to rule without

The results of the comparison are mostly equal to those of the comparison above. With search depth 1, the random projection is outperformed in every category. The biggest difference is in the category ”Views Predicted”. With search depth 2 however, it is the other way around. Random projection outperforms the Conditional Function in every category, even ”Views Predicted”, although only slightly. The latter is different to the results of the other comparison. Looking at random projection, an increase in history length from 1 to 2 does increase the performance further, while the Conditional Function without random projection decreases in performance with a longer history length.

Summary
Random projection can fully exploit the increase of history length and does not decrease in performance with an increase in search depth as substantially as rules without random projection. They therefore outperform rules without random projection when using a search depth of 2. The results for search depth 1 however are not as good as the results of rules without random projection, which means the overall best results are still achieved by rules without random projection. Therefore, methods without random projection are the better choice.

We now analyse the four methods from the last section regarding the impact of the history length. We look at the View rule, the Conditional Function, the View rule with random projection and the Conditional Function with random projection.

Depth 1

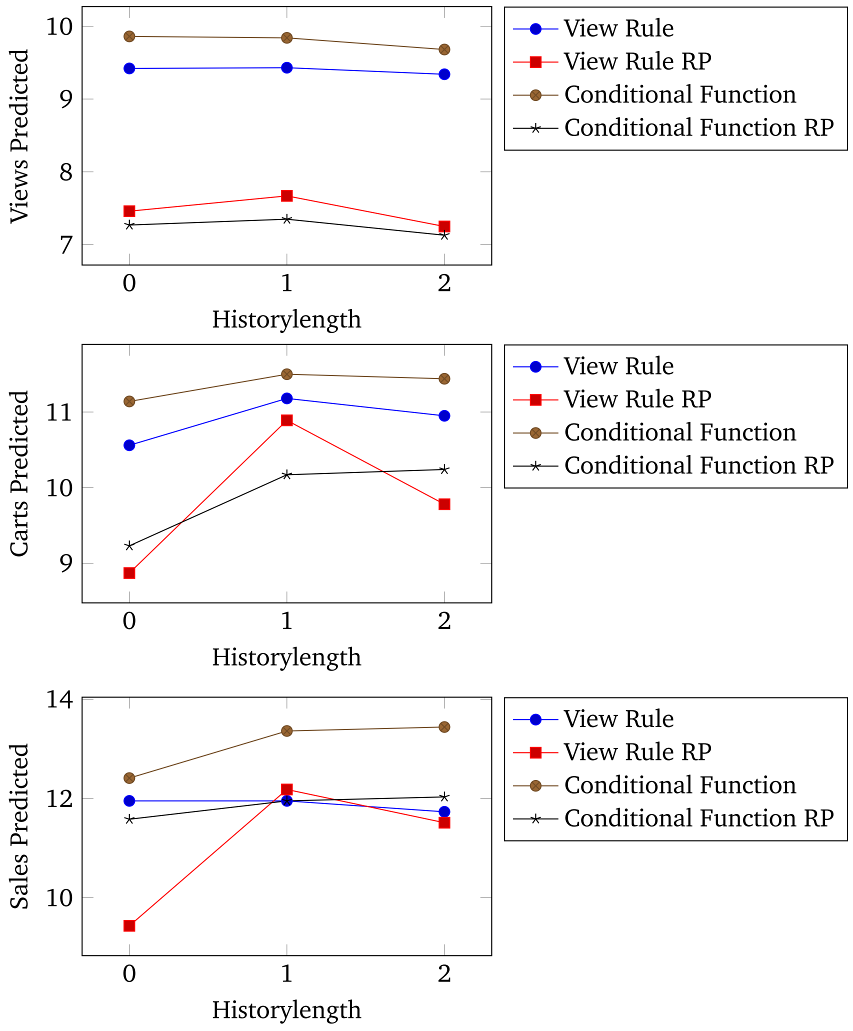


Figure 5.27: Impact of history length on rules with search depth 1

In the category ”Views Predicted”, the history length does not have much impact. It seems that only the current item is important to predict which item a user clicks on next. In the categories ”Carts Predicted” and ”Sales Predicted”, a history length of 1 does best for simple rules. A longer history length leads to a decrease in performance. This may be due to the fact that about 25 percent of all session only have a length of 2 or less. This means, a history length of 2 is never needed one fourth of the time, and on the other hand there may not be enough training examples in the small train set for a good use of history length 2. Conditional Functions further improve with history length 2 in the category ”Sales Predicted”. Methods with random projection are more sensitive to the history length than methods without random projection. The View rule with random projection profits the most from the increase of the history length from 0 to 1.

Depth 2

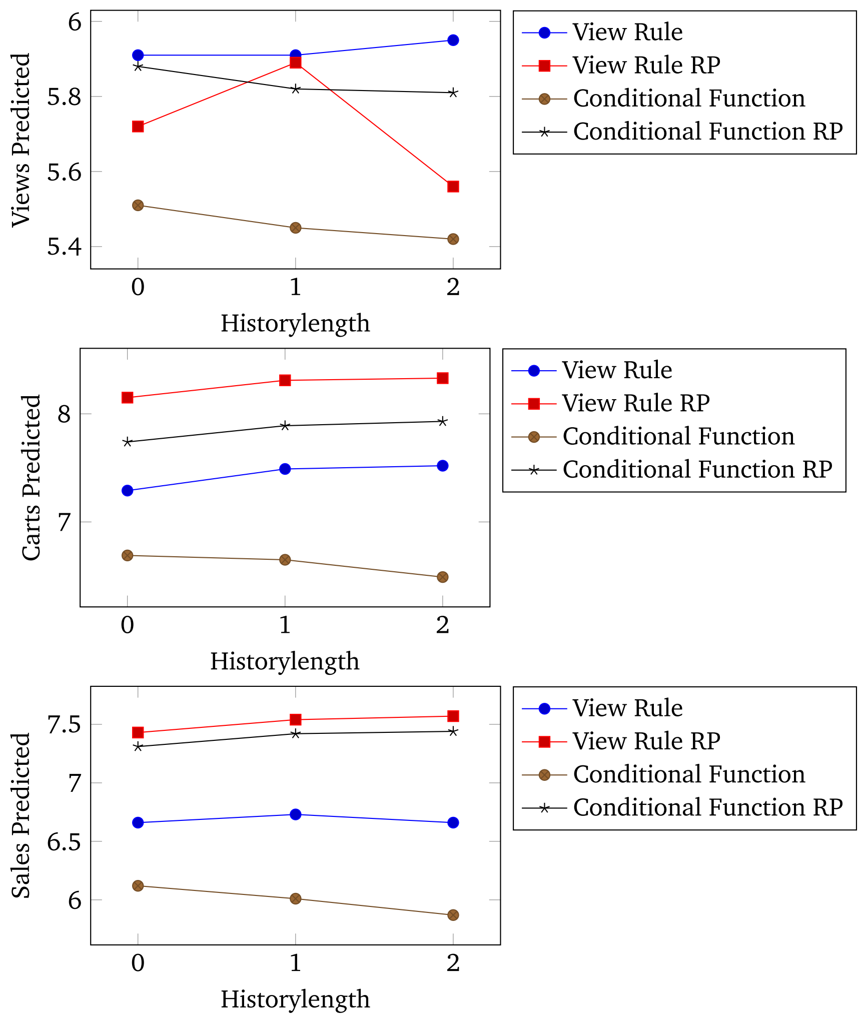


Figure 5.28: Impact of history length on rules with search depth 2

In all categories, the history length does not have much impact. It seems that with search depth 2, the history length cannot be utilised as much as there is more room for false predictions (see next section for further explanation). The only exception is the View rule with random projection, where an increase is noticable from 0 to 1, and a substantial drop from 1 to 2. In the category "Carts Predicted" and "Sales Predicted", there is a difference between the Conditional Function and the other rules. The Conditional Function steadily decreases in performance. Because it generally performs worst with search depth 2, it also cannot make use of the history length in any way. With random projection on the other hand, history length 2 performs best in the category "Carts Predicted" and also "Sales Predicted". 25 percent of all session have only 2 events or less, but most of these sessions only consist of view events. Sessions with cart and sale events are longer, and random projection seems to fully make use of the history length there. Methods without random projection on the other hand cannot make much use of a longer history length.

5.4.6 Impact of Search Depth

We now analyse the four methods regarding the impact of the search depth. We look at the View rule, the Conditional Function, the View rule with random projection and the Conditional Function with random projection. We use history length 1 for comparison, because most methods do best with that history length.

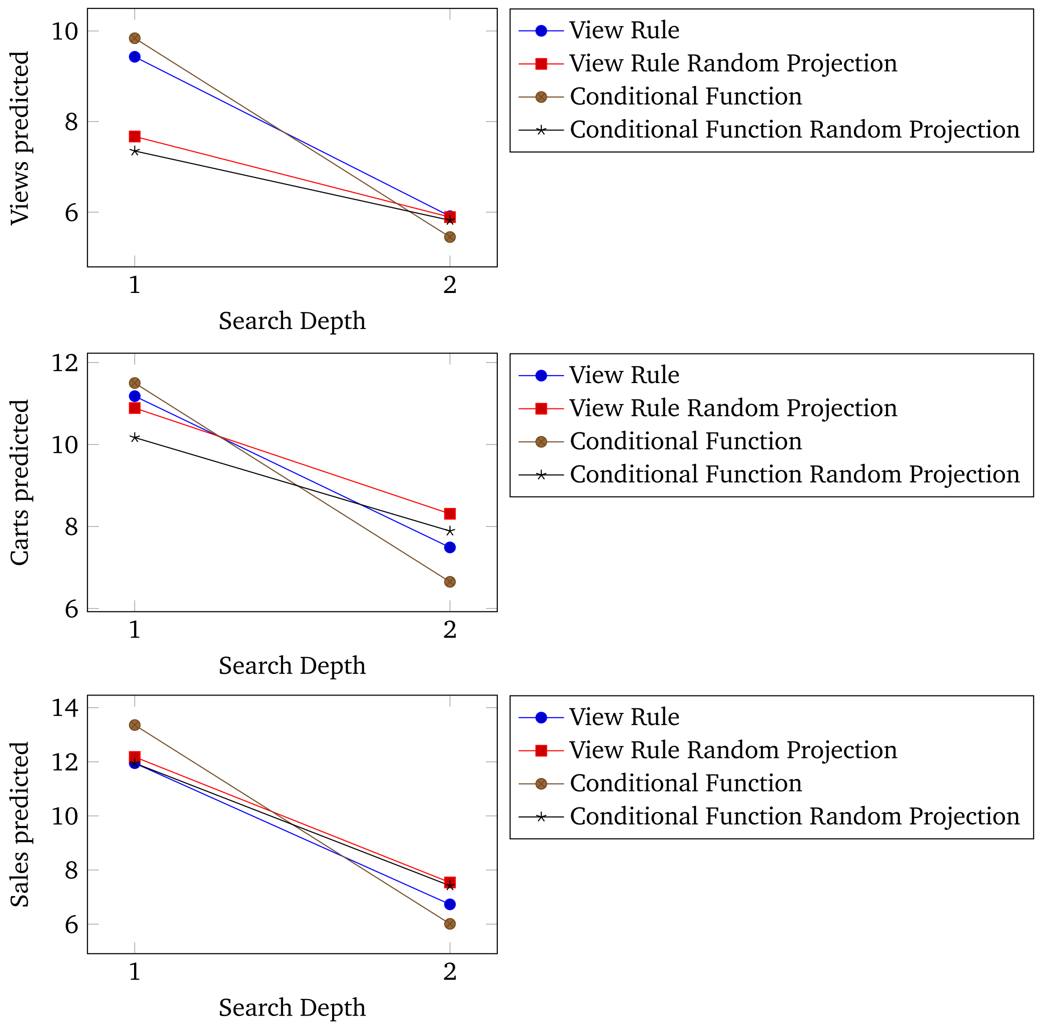


Figure 5.29: Impact of search depth on rules. History length 1 used for comparison

An increase of the search depth leads to a drop in all categories. A reason is the low increase of the maximum reachable scores for search depth 2, compared to search depth 1. For the category "Views Predicted" for example, we can predict a maximum of 41.21 percent of all view events with search depth 1. With search depth 2, this score increases only by about 15 percent to 54.27 percent. At the same time, the number of items the rules have to narrow down to three recommendations rises from an average of 120 to an average of 794 items. This means, that for search depth 2, the ratio of "good" items and "bad" items is much worse than for search depth 1. The chance to predict a view is therefore higher with search depth 1, as many "bad" items are not considered. The same reasoning can be applied to the other two categories.

The Conditional Function drops the most in performance. A reason for this is that every Conditional Function consists of three rules, which each have to narrow an average of 794 items down to 1 recommendation. It seems that this yields too many errors in comparison to search depth 1, where on average only 120 items are taken into account. Simple rules on the other hand only have to narrow down those items down to 3 recommendations once, which is why the drop in performance is not as substantial.

5.4.7 Summary

We first compared the rules described in section 4.1. We compared them for search depth 1 and 2, and did this for a history length of 0, 1 and 2. We found that overall, the View rule trained with Hoeffding Trees does best. The View rule outperforms the baseline in the categories ”Carts Predicted” and ”Sales Predicted”, but not in the category ”Views Predicted”. In this category, it seems that a simple popularity measure of an item performs best.

After that, we looked at the Conditional Function. We first identified the View rule as the best method to predict one recommendation that will most likely be sold, and then tested the resulting six combinations. The results show that the Conditional Function with ”View / Conditional Cart No Sale / Conditional View No Cart Sale” does best. We then compared this Conditional Function with the best result of our first tests, which is the View rule. The Conditional Function outperforms the View rule, but only by a slight margin. We also found out that the Conditional Function does decrease more substantially in performance from an increase in search depth.

We then tried to further improve the results of the Conditional Function and the View rule by using random projection. The results show that there is an improvement regarding search depth 2, in which rules with random projection perform better compared to rules without random projection. But the overall best results results are still achieved with search depth 1 and therefore by rules without random projection.

We then examined the four methods - View rule and Conditional Function, both with and without random projection - regarding the impact of history length and the impact of search depth. For all rules without random projection, history length 1 is the best option, for rules with random projection, history length 2 further increases the performance. An increase in search depth leads to a decrease across all categories. This is due to the worse ratio of ”good” and ”bad” items: The number of items the rules have to narrow down to three recommendations rises from an average of 120 to an average of 794 items, while the maximum possible score for predicting views, carts and sales rises only by about 15 percent. The drop in performance is not as substantial for rules with random projection compared to rules without.

5.4.8 Determining the Top 2 in Every Measured Category

We now determine the best two methods each for the categories ”Views Predicted”, ”Carts Predicted” and ”Sales Predicted”. We take into account all simple rules trained with Hoeffding Trees, the best Conditional Function and rules enhanced with random projection. We also take into account the baseline, but will not use it for the best two methods. We leave out the simple rules trained with Naive Bayes because of their bad performance. For every method, its best performance for each category is taken into account, regardless of search depth and history length. For example, the best result of the View rule in the category ”Sales Predicted” is with the parameters ”History length 1, Search Depth 1”, while for ”View Rule RP” it is with the paremeters ”History length 2, Search Depth 1”. Following tables show the results. They are ordered, with the best result at top. The abbreviations in the third column show for which parameters the best result was achieved.

Method name	Views Predicted	Parameters
Baseline	10.09	D1
Cond Func	9.86	HL0, D1
View Rule	9.43	HL1, D1
View+Sale Rule	8.81	HL0, D1
View+Cart Rule	8.68	HL1, D1
Sale Rule	7.9	HL0, D1
Cart Rule	7.79	HL1, D1
View Rule RP	7.46	HL1, D1
Cond Func RP	7.27	HL1, D1

Method name	Carts Predicted	Parameters
Cond Func	11.5	HL1, D1
View Rule	11.18	HL1, D1
View+Cart Rule	10.98	HL2, D1
View Rule RP	10.89	HL1, D1
Cart Rule	10.54	HL1, D1
Cond Func RP	10.24	HL2, D1
Baseline	10.09	D1
View+Sale Rule	10.01	HL1, D1
Sale Rule	9.62	HL2, D1

Method name	Sales Predicted	Parameters
Cond Func	13.44	HL2, D1
View+Cart Rule	12.77	HL2, D1
View Rule RP	12.18	HL1, D1
Cond Func RP	12.03	HL2, D1
View Rule	11.95	HL1, D1
Cart Rule	11.95	HL1, D1
View+Sale Rule	11.14	Hl1, D1
Baseline	10.07	D1
Sale Rule	9.57	HL0, D1

Table 5.2: Ranking of all rules in the categories ”Views Predicted”, ”Carts Predicted” and ”Sales Predicted”

In the category ”Views Predicted”, the baseline performs best. The two best methods are Conditional Function and View rule. In the category ”Carts Predicted”, the View Rule and the Conditional Function perform best, too. In the category ”Sales Predicted”, the Conditional Function and the View+Cart rule perform best. Therefore, we will use the three methods ”View rule”, ”Conditional Function” and ”View+Cart rule”, all trained with Hoeffding Trees, for further analysis and for evaluation on the big testset.

5.5 Results for Big Testset

In the last section, we thoroughly tested all methods and used the measures ”Views Predicted”, ”Carts Predicted” and ”Sales Predicted” to compare them. For each measure, we made a ranking and picked the two best performing rules which will be tested on the big test set now. Those methods are ”View rule”, ”View+Cart rule” and ”Conditional Function” with the combination ”View / Cart No Sale / View No Cart Sale”. We first directly compare them with regards to the history length and the search depth. We then analyse the impact of search depth and history length. We also compare the results of the big, i.e. validation test set with the results of the small, i.e. tuning and analysis test set.

5.5.1 Baseline and Maximum Possible Scores

The maximum possible scores are computed by returning all items as a recommendation, that are found in the first step of the algorithm (finding all items with a difference by one feature). On average, 122 items are found with one iteration, 803 with two. Following scores are achievable:

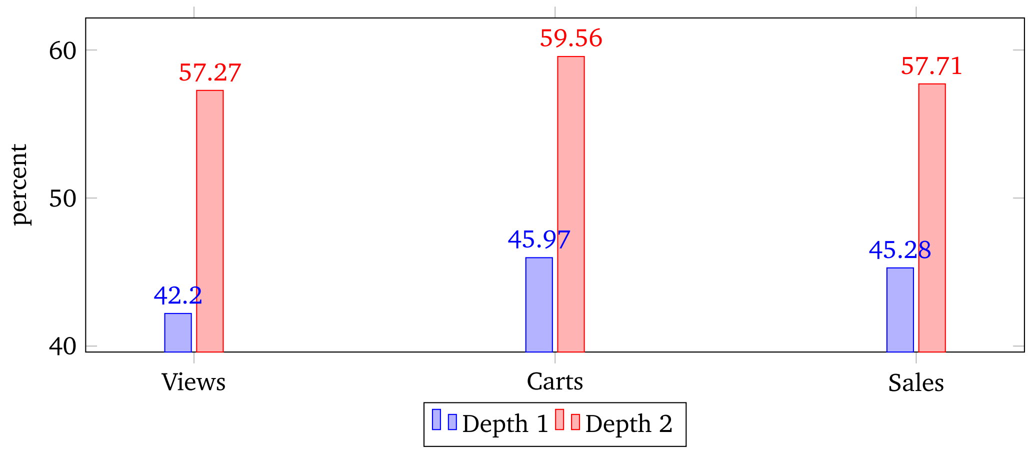


Figure 5.30: Maximum Possible Scores for Search Depth 1 and 2

Returning all items obviously is not a suitable baseline, because we must not recommend more than three items. Therefore the baseline is the top three of the popularity count of the items. Every view, cart and sale event is used to compute the score. Example: Item A is viewed 10 times, carted 1 time and sold 0 times. Item B is viewed 7 times, carted 4 times and sold 1 time. Item A then has a score of 10+1+0=11, item B has a score of 7+4+1=12. The higher the score, in other words the more popular an item is, the higher it is ranked. We chose this baseline because it is independent of any other parameters like history length. Essentially, we just recommend items that are popular in general. Following scores are achieved:

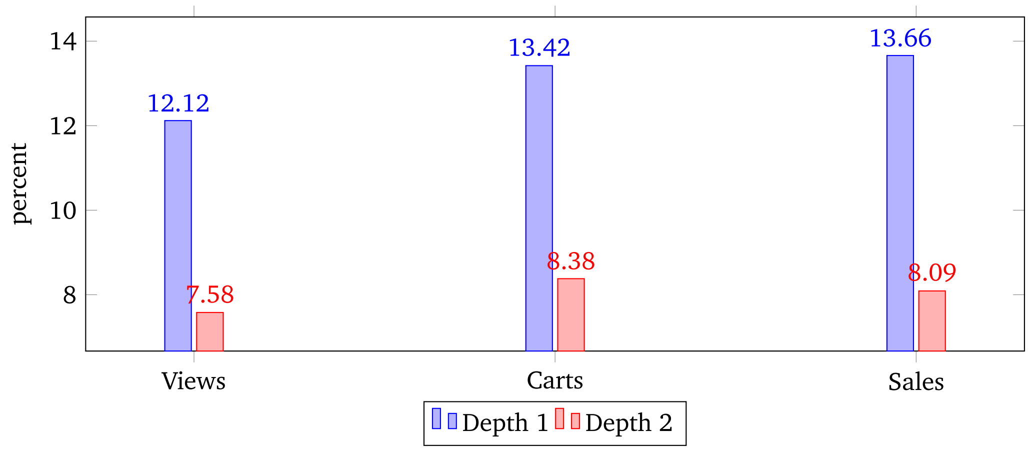


Figure 5.31: Achievable scores in the measures ”Views Predicted”, ”Carts Predicted” and ”Sales Predicted” for the baseline

5.5.2 Direct Comparison

History length 0

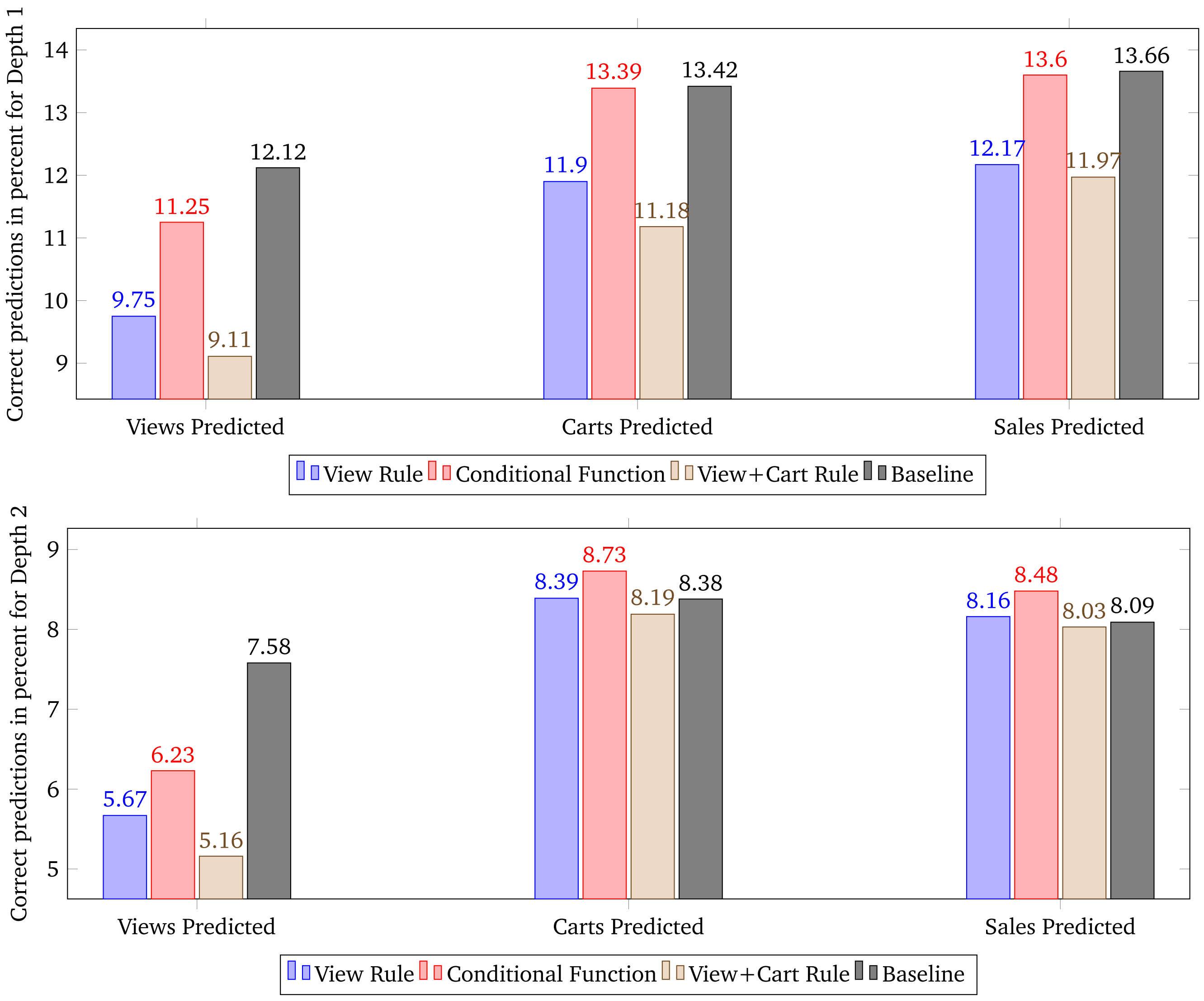


Figure 5.32: Scores of the rules for history length 0

For search depth 1, all rules are inferior to the baseline. The best of the remaining methods is the Conditional Function. It outperforms all other methods by 1 to 2.5 percent in every category and is very close to the baseline in the categories ”Carts Predicted” and ”Sales Predicted”. The View+Cart rule performs worst in every category, the View rule is slightly better.

For search depth 2, the baseline is best at predicting views. Out of the other four rules, the Conditional Function does best in this category. The Conditional Function is also the only method to surpass the baseline in the other two categories by more than a slight margin. In contrast to search depth 1, the measurements deteriorate almost by 5 percent for the baseline and the Conditional Function. The drop in performance for the other rules is not as substantial, but they still perform worse than the Conditional Function.

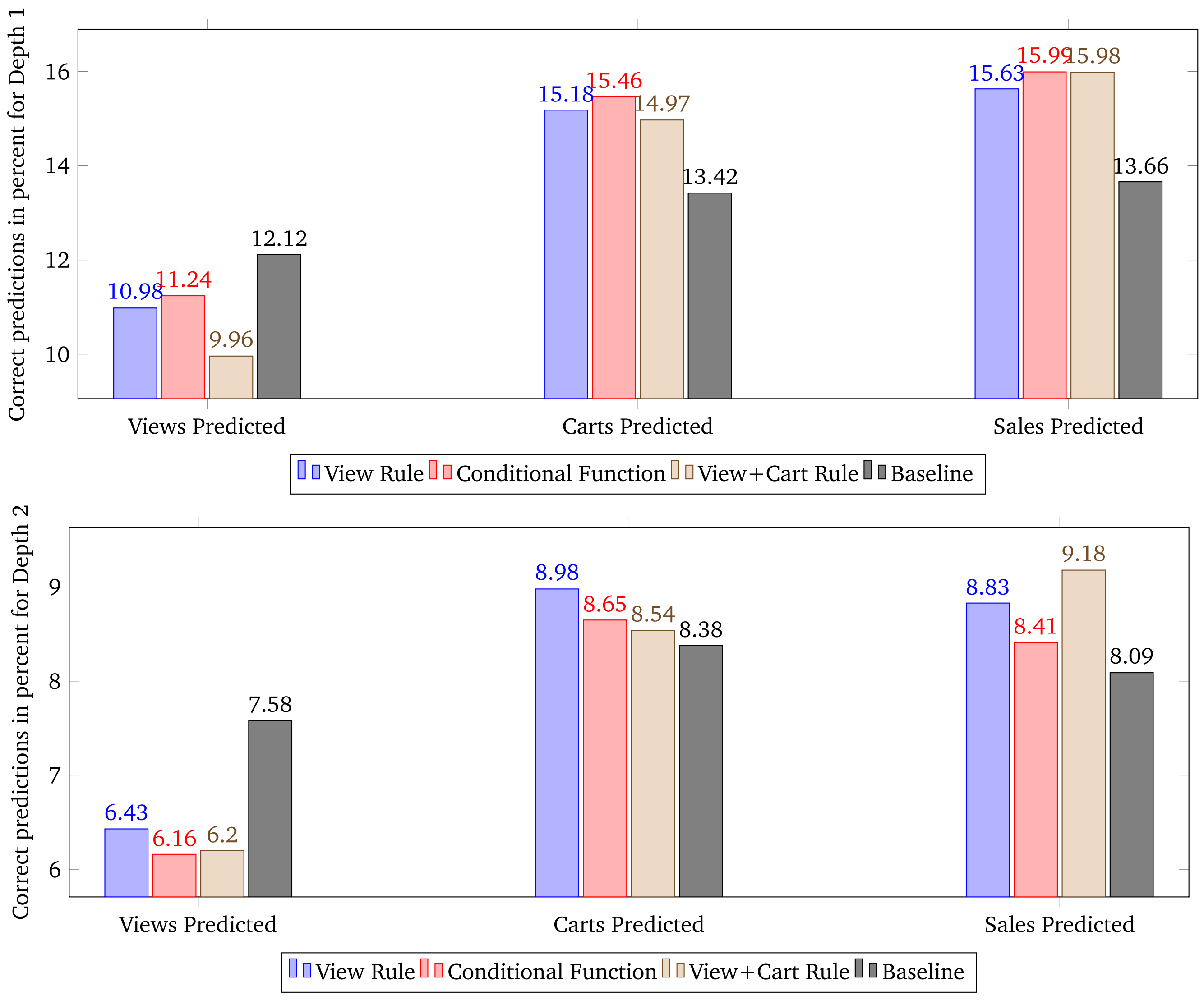
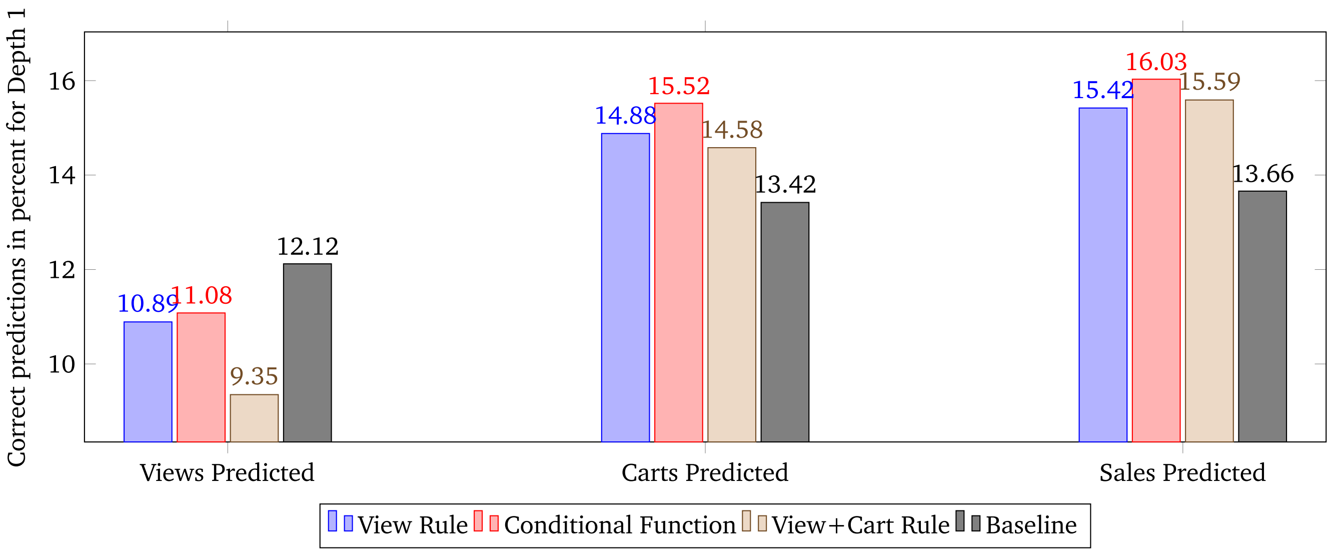


Figure 5.33: Scores of the rules for history length 1

For search depth 1, each of the rules increase substantially. The View rule performs second best in the category ”Carts Predicted” with a score of 15.18. The View+Cart rule performs second best in the category ”Sales Predicted” with a score of 15.98. The Conditional Function performs best in both these categories, but only by a slight margin. Compared to history length 1, search depth 1, the simple rules improve substantially, especially in the categories ”Carts Predicted” and ”Sales Predicted” with a plus of 4 and 3.5 percent. All rules are still inferior to the baseline for the measure ”Views Predicted”.

For search depth 2, the baseline is best at predicting views. Out of the other four rules, the View rule does best in this category, though they are all very close to each other. In the categories ”Carts Predicted” and ”Sales Predicted”, the simple rules improve substantially in comparison to history length 0 and now surpass the baseline. The View+Cart rule is now best at predicting sales, while the View rule is best at predicting carts. The Conditional Function performs worse now in comparison to history length 0.

History length 2



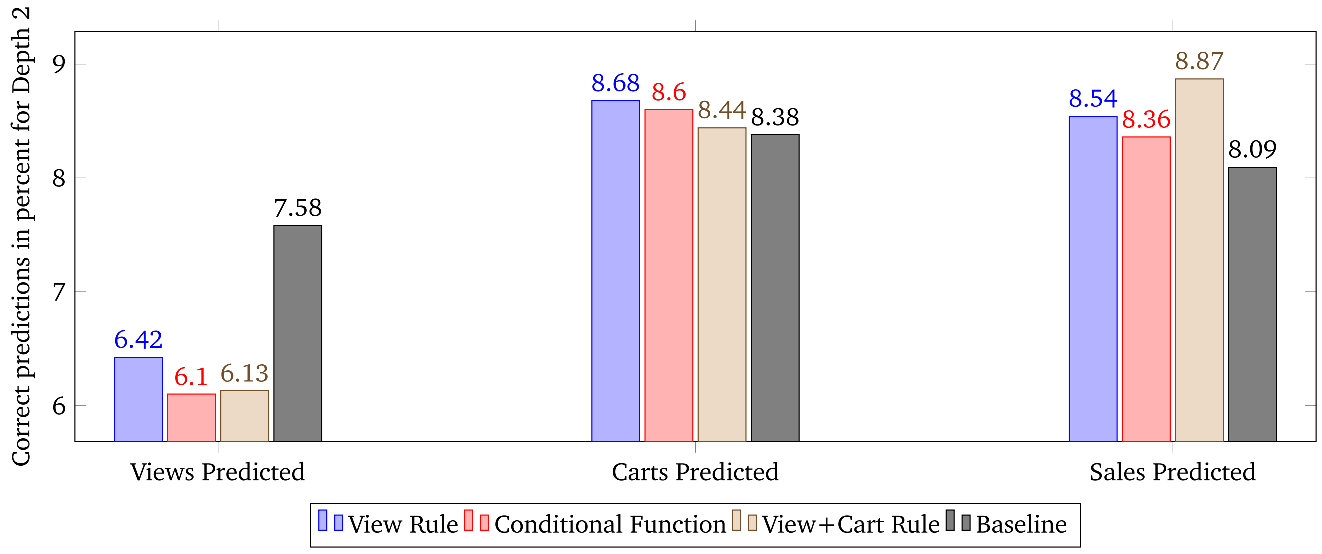


Figure 5.34: Scores of the rules for history length 2

For search depth 1, each of the rules again perform better than the baseline in the categories ”Carts Predicted” and ”Sales Predicted”, but worse in the category ”Views Predicted”. The View rule performs second best in the category ”Carts Predicted”,the View+Cart rule performs second best in the category ”Sales Predicted”. Compared to history length 1, search depth 1, the simple rules decrease slightly, but they are still better than with history length 0. The Conditional Function on the other hand further improves in the categories ”Carts Predicted” and ”Sales Predicted”, although only very slightly. The Conditional Function overall performs best out of all the three rules.

For search depth 2, the baseline is best at predicting views. Out of the other four rules, the View rule does best in this category, though they are all very close to each other. In the categories ”Carts Predicted” and ”Sales Predicted”, the simple rules decrease slightly in comparison to history length 1, but still surpass the baseline. The View+Cart rule is again best at predicting sales, while the View rule is best at predicting carts. The Conditional Function performs worse in comparison to history length 1 and 2. Overall, the results of all rules slightly decreased in all categories compared to the results of history length 1.

Summary

For the measure ”Views Predicted”, the baseline performs best with 12.12 percent. The Conditional Function is second with 11.24 percent. The View rule and the View+Cart rule perform best with history length 1, the Conditional Function performs best with history length 2. The simple rules benefit more from an increase of the history length from 0 to 1. An increase of the search depth always leads to a drop in performance. We will elaborate on possible reasons in sections 5.5.4 and 5.5.5.

5.5.3 Comparison to Results of Small Testset

For each method, we compare the best results of the small testset with the best results of the big testset, regardless of history length and search depth. For example, the View rule might perform best with history length 1 and search depth 1 in the category ”Views Predicted”, while performing best with history length 1 and search depth 2 in the category ”Sales Predicted”. We compare the results for the measures ”Views Predicted”, ”Carts Predicted” and ”Sales Predicted”. We normalise both the results from the small and big test set. This is better for comparison, because for example a score of 10 percent in the category ”Views Predicted” is much better if there is a maximum achievable score of 40 percent possible compared to a maximum achievable score of 50. The normalisation is done by converting the results of the big test set to the maximum achievable scores of the small test set. Example: The maximum achievable score for ”Views Predicted” with search depth 1 is 41.21 percent on the small test set, and 42.2 percent on the big test set. A ”Views Predicted”-score of 10 on the big test set is then normalised to $10 \times \frac{41.21}{42.2} = 9.77$.

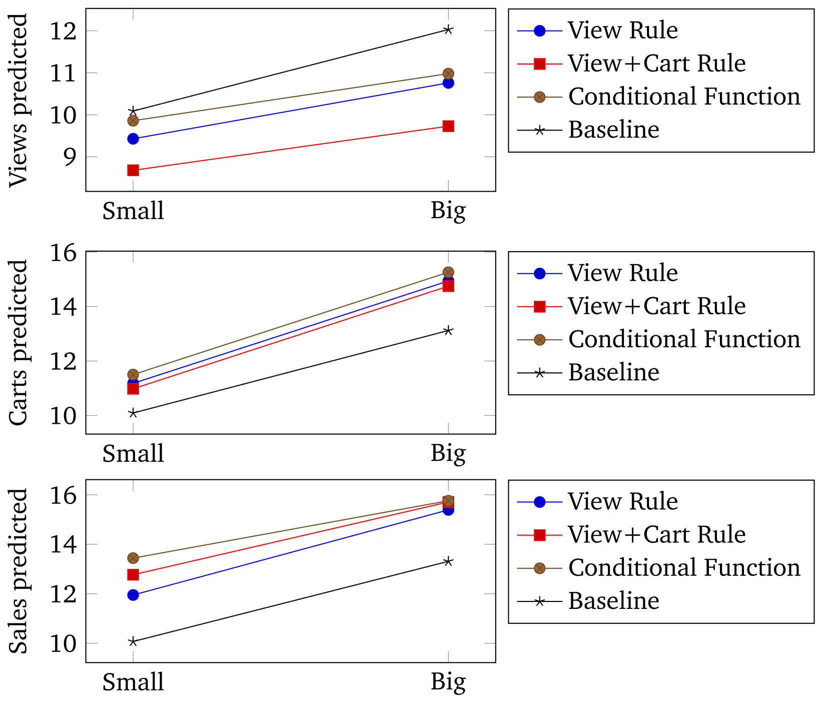


Figure 5.35: Results of best rules on small set compared to results on big set

Interestingly, the results stay mostly the same. One would expect that things like summer and winter collections make an impact on the big test set, but this is not the case. Across all categories, an improvement is noticable. In the category ”Sales Predicted”, the results for the Conditional Function stay mostly the same. Simple rules on the other hand increase more. While they were worse than the Conditional Function in the categories ”Carts Predicted” and ”Sales Predicted” on the small testset, they are now almost equal to the Conditional Function on the big test set. The baseline improves most in the category ”Views Predicted”, which is expected due to the higher number of training data, that are directly used for product counts, that show the item popularity. Overall, more training instances lead to better results.

5.5.4 Impact of History Length

We now analyse the best methods regarding the impact of history length. The results of the analysis of the big test set are mostly the same compared to those of the small test set, but are still discussed here in full for better readability.

Depth 1

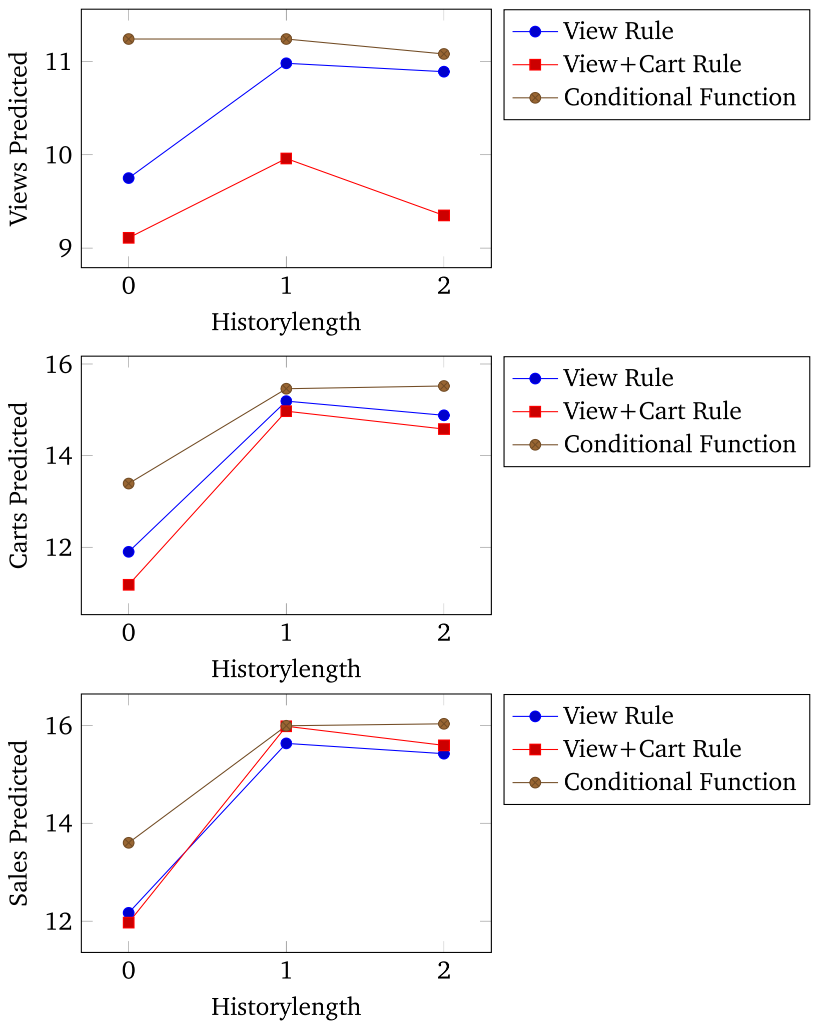


Figure 5.36: Impact of history length on rules with search depth 1

For the measure ”Views Predicted”, history length 1 performs best for all rules. A history length of 2 is not desirable, it performs worse than history length 1 for the View and the View+Cart rule and worst for the Conditional Function. For the measures ”Carts Predicted” and ”Sales Predicted”, the trend is equal for the simple rules. What is different is the amount of increase. The View and View+Cart rule experience a substantial increase and are almost as good as the Conditional Function, but drop in performance for history length 2. The Conditional Function on the other hand already performs quite well with history length 0 and does not increase as much as the other two rules with history length 1. There also is a slight increase from history length 1 to 2. It seems the Conditional Function can make full use of the history length.

Depth 2

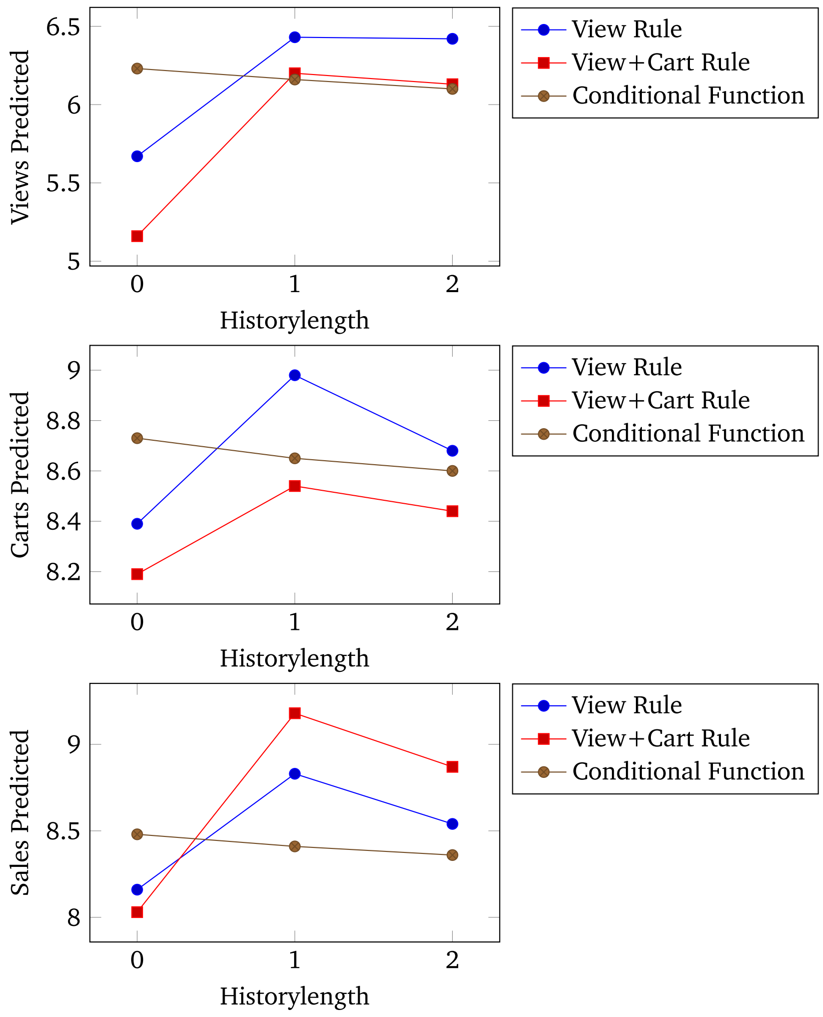


Figure 5.37: Impact of history length on rules with search depth 2

The Conditional Function performs best with a history length of 0, its performance steadily decrease with a longer history length. The View and View+Cart rule on the other hand both experience a substantial increase in performance when using a history length of 1: While they are inferior to the Conditional Function with history length 0, they are superior with history length 1. For both, the performance drops with history length 2. Their trend is therefore equal to that of search depth 1.

5.5.5 Impact of Search Depth

We now analyse the best methods regarding the impact of the search depth. We look at the View rule, the View+Cart rule and the Conditional Function. We use history length 1 for comparison, because most methods perform well with that history length. The results of the analysis of the big test set are mostly the same compared to those of the small test set, but are still discussed here in full for better readability.

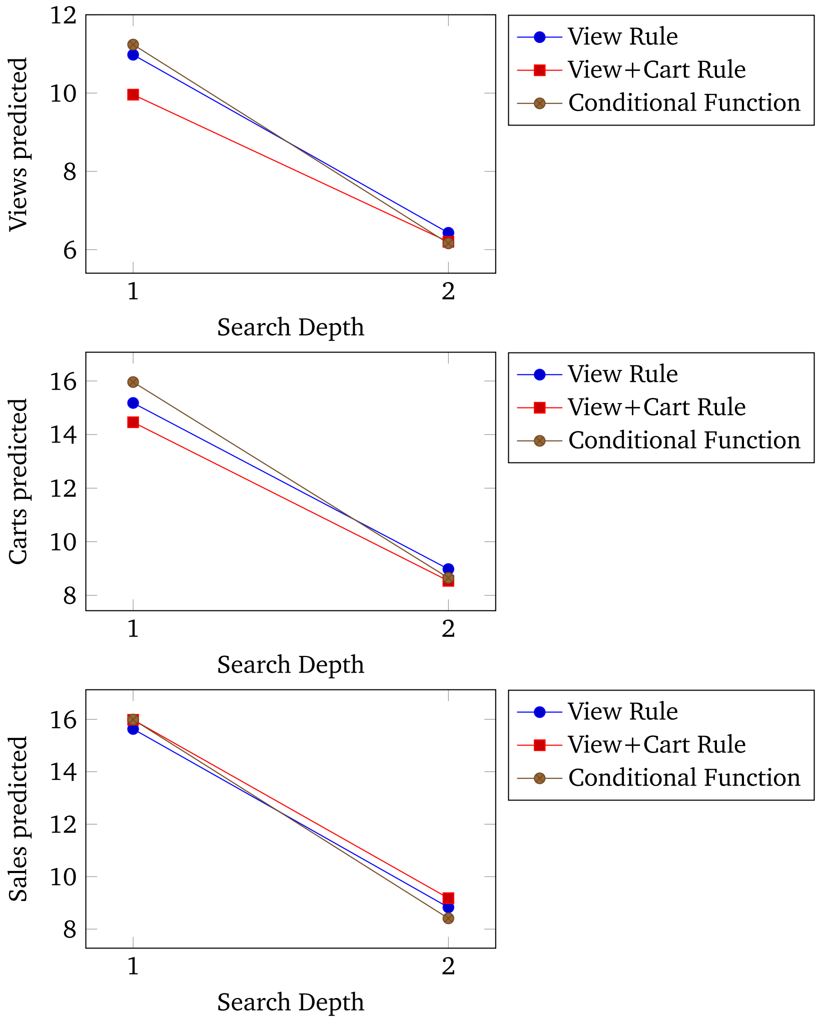


Figure 5.38: Impact of search depth on rules. History length 1 used for comparison

An increase of the search depth leads to a drop in all categories. A reason is the low increase of the maximum reachable scores for search depth 2, compared to search depth 1. For the category "Views Predicted" for example, we can predict a maximum of 42.2 percent of all view events with search depth 1. With search depth 2, this score increases only by about 15 percent to 57.27 percent. At the same time, the number of items the rules have to narrow down two three recommendations rises from an average of 122 to an average of 803 items. This means, that for search depth 2, the ratio of "good" items and "bad" items is much worse than for search depth 1. The chance to predict a view is therefore higher with search depth 1, as many "bad" items are not considered. The same reasoning can be applied to the other two categories.

The ranking of performance stays almost the same. In every category, the Conditional Function experiences the most substantial decrease. While it performs better than the other two rules with search depth 1, it performs worst with search depth 2. It seems that it is too hard to narrow down on average 803 items down to 1 recommendation, which the Conditional Function has to do three times to make its three recommendation. Simple rules on the other hand have to narrow down these on average 803 items down to only 3 recommendations one time, hence the drop in performance for simple rules is not as big in comparison to the Conditional Function.

5.5.6 Summary

On the small set, we identified the best methods and validated their results on the big set. The methods analysed are the View rule, the View+Cart rule and the Conditional Function.

The results of the big testset are in line with the results of the small testset. The ranking of the rules in terms of performance has not changed. The Conditional Function still performs best out of all rules, but the View rule and View+Cart rule both improved more substantially than the Conditional Function. Compared to the small testset, improvements were noticeable in every category, which is expected due to the higher number of training examples. The table below shows the best result of each rule, regardless of search depth and history length.

Method Name	Views	Carts	Sales	Median Rank
Baseline	1 (12.12)	4 (13.42)	4 (13.66)	4
View Rule	3 (10.98)	2 (15.18)	3 (15.63)	2
View+Cart Rule	4 (9.96)	3 (14.97)	2 (15.98)	3
Cond Func	2 (11.25)	1 (15.52)	1 (16.03)	1

Table 5.3: Overall best performance and ranking of rules in the categories "Views Predicted", "Carts Predicted" and "Sales Predicted"

6 Conclusion

In this master thesis, we discussed the utilisation of preference-based learning for recommender systems, specifically the task of ranking a set of items using a training set consisting of pairwise preferences. We introduced three core rules which we derived from the events of the dataset:
1. The View rule, which says A > B if item A is viewed after item B.
2. The Cart rule, which says A > B, where A is an item that is carted, and B is an item, which was not carted in that user session.
3. The Sale rule, which says A > B, where A is an item that is sold, and B is an item, which was not sold in that user session.
Deriving from those core rules, we created five simple rules, namely ”View”, ”Cart”, ”Sale”, ”View+Cart”, ”View+Sale”, where the last two are combinations of the first three. We also introduced a set of Conditional Functions, which are several rules combined where pairwise preferences depend on other items. Such a rule is for example the ”Cart No Sale” rule which says (A > B | C), where A is an item that is carted, but not sold, B is an item that was only viewed, and C is an item that was sold in that session. The reasoning behind these Conditional Functions is that we want to recommend items depending on the potential choice of the user not to click on a recommendation: We first recommend an item, that will most likely be sold later. If the user does not click on that item, we recommend another item. The choice of the second and third item depend on the first and second item, respectively.

We then analysed the proposed methods using three measures: ”Views Predicted”, which shows how many views the rule predicts, ”Carts Predicted”, which shows how many carts the rule predicts and ”Sales Predicted”, which shows how many sales the rule predicts. For the last two measurements, it is not important that a view and consequently a cart/sale event of the item recommended comes directly after the recommendation, but that the recommended item is carted or bought at some point later in the session. Due to the dataset being ”cold” and not having the possibility to test the recommender system in a live environment, we cannot know how the user would have reacted, so we assume a hit if the user carts or buys the item sometime later, because the user could have taken a shortcut.

We first tested thoroughly on a small chunk of the whole dataset, about 0.5 percent of it. We trained on the first 90 percent of the set, and tested on the remaining 10 percent. We tested each of the five simple rules with a model trained with Naive Bayes, and a model trained with Hoeffding Trees. The results of Naive Bayes are inferior to those of Hoeffding Trees. The results for Hoeffding Trees look as following:
Views Predicted: The View rule does best. This is expected due to the way the rule works. It learns on view events, so it should also predict view events best. However, the best rule is still inferior to the baseline. An explanation for this is that a simple popularity measure might give better recommendations to many users, as they are interested at what is popular or trending at the moment.
Carts Predicted: The view rule again does best. One would expect the Cart rule to perform best here, but it seems that users are making too diverse carts, and there may not have been enough training examples yet. Also, due to the algorithm only searching for items that differ in one or two features, it might be that items which are more likely to get carted are not included, because they are ”too far away” from the current item. Because of this, a more conservative approach in the form of the View rule works better.
Sales Predicted: In this category, the View+Cart rule does best, with the View rule in close second. The Sale rule does not perform very well, which might be due to the low amount of training examples as well as users making too diverse sales. Cart events on the other hand can be utilised, which is why the View+Cart is slightly ahead of the View rule. This however shows, that the conservative approach of learning from view events lays the groundwork.

After the simple rules, we tested the Conditional Functions. In order to accomplish this, first the best simple rule for predicting sales with only one recommendation had to be found. An analysis showed that it is the View rule. With the View rule making the first recommendation, all six combinations for forming a Conditional Function were tested. It turned out that the Conditional Function with ”View / Cart No Sale / View No Cart” performs best. The Conditional Function was then compared to the best results of the simple rules. For search depth 1, the Conditional Function always performed better, for search depth 2, it always performed worse. An explanation for this is that it is too hard to narrow on average 794 items down to 1 recommendation, which the Conditional Function has to do three times to make its three recommendation.

No further improvements can be made if the rules are preprocessed with random projection, that is, the number of features is projected onto a lower dimension. A projection down to 300 features showed the best results in a first quick test.

Random projection was applied to both the best simple rule and the Conditional Function. For search depth 2, the results show an increase in the categories ”Carts Predicted” and ”Sales Predicted”. For search depth 1, the results are inferior to the simple rules. It seems that rules enhanced with random projection are especially good at recommending items that are not too similar to the current item. This may be due to the fact that the training examples often include pairwise preferences where the items are very different from each other and with a search depth of 2 the learned model is more accurate. Overall however, random projection performs worse than the simple rules, since the best results are achieved with search depth 1 and random projection performs worse there.

After the first tests, we identified the top two methods in each measure, which are three methods, namely the View rule, the View+Cart rule and the Conditional Function. These three methods were used for further evaluation and validation on the big set. Due to more training examples, the results are better than on the small trainset. However, the baseline still performs best in the category ”Views Predicted”. It seems that a simple popularity measure for ranking suits the exploratory style of browsing items best. The ranking of the rules in terms of performance has not changed. The Conditional Function still performs best out of all rules, but the View rule and View+Cart rule both improved more substantially than the Conditional Function.

An analysis of the impact of search depth and history length showed, that the best results are achieved with a history length of 1 and a search depth of 1 for the simple rules. A larger history length leads to a decrease in performance. This may be due to the fact that about 25 percent of all session only have two events or less. This means, a history length of 2 is never needed one fourth of the time, and on the other hand there may not be enough training examples in the small trainingsset for a good use of history length 2. The Conditional Function seems to not be affected by this due to its more complex structure and improves further with a history length of 2, but only very slightly. Why a search depth of 2 is generally worse can be explained with an example: With search depth 1, we can predict a maximum of roughly 40 percent of all view events. With search depth 2, this score increases by only about 15 percent to roughly 55 percent. At the same time, the number of items the rules have to narrow down to three recommendations rises from an average of about 120 to an average of roughly 800 items. The maximum achievable scores in the other two categories are similar. This means, that for search depth 2, the ratio of ”good” items and ”bad” items is much worse than for search depth 1.

Method Name	Views	Carts	Sales	Median Rank
Baseline	1 (12.12)	4 (13.42)	4 (13.66)	4
View Rule	3 (10.98)	2 (15.18)	3 (15.63)	2
View+Cart Rule	4 (9.96)	3 (14.97)	2 (15.98)	3
Cond Func	2 (11.25)	1 (15.52)	1 (16.03)	1

Table 6.1: Overall best performance and ranking of rules in the categories ”Views Predicted”, ”Carts Predicted” and ”Sales Predicted”

Different methods are best for different catogories. If maximising the predicted views is important, the baseline is the best choice. If the prediction of carts and sales need to be maximized, the Conditional Function performs best, but both simples rules are only sligthly worse.

7 Outlook

While we have extensively discussed and analysed the problem of applying preference-based learning to recommender systems by learning from pairwise preferences, we have not focussed much on feature engineering, classifier selection and parameter tuning as this would have taken too much time and would go beyond the scope of this thesis. Quite certain, there are better results possible on the Zalando dataset and the discussed approach, if said things were done. For example, user features are missing completely, and therefore an explicit user model is missing. Features like the favorite colour of a user or the favourite brand are just a few things to name here. One also has to note that the Zalando dataset only contains a subset of all features Zalando uses. Additional features, for example features that describe the look of the items in more detail, would certainly help. Second, the influence of seasons was ignored. Click logs from the winter season might not be suitable for learning recommendations for the summer season. Also, we only tested with the classifiers Naive Bayes and Hoeffding Trees. The latter was not tuned extensively, as this was not the focus of the thesis. Another improvement could be made with the algorithm that is used to preselect a number of items that are then narrowed down to three recommendations by the rules. The algorithm selects all adjacent items of the current item where two items are adjacent if they differ by exactly one feature. Another metric to decide if two items are adjacent, for example by employing collaborative filtering methods, could be better.

The results might also be skewed, because at the start of a session, many users might have clicked on items chosing from an overviewlist. This means that they never looked at many recommendations in the first place, so the real success rate might be lower. However, the results presented are still promising. Deriving pairwise preferences from implicit feedback is much more natural than creating explicit numeric ratings from implicit feedback. Yet there are many possibilities on how to construct the rules for pairwise preference creation. In this thesis, we introduced three core rules deriving from view, cart and sale events and used them to create many variants of rules. Other and possibly better rules are left to be discovered.

Lastly, we did not take into account factors such as context or diversity of recommendations. These factors have to be taken into account, too, when creating a robust recommender system. The measures used in this approach, namely ”Views Predicted”, ”Carts Predicted” and ”Sales Predicted” might not be the key measures others want to optimise. Measures like Return On Investment (ROI) might be important, too.

Acknowledgment

We want to thank Zalando for providing us the dataset which is an integral part of this thesis.

References

[1] Salton, G. Automatic Text Processing. Addison-Wesley, 1989.
[2] Billsus, D. and M. Pazzani. User modeling for adaptive news access. User Modeling and User-Adapted Interaction, 10(2-3):147-180, 2000.
[3] Gediminas Adomavicius and Alexander Tuzhilin. Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions, 2005.
[4] Claypool, M., A. Gokhale, T. Miranda, P. Murnikov, D. Netes, and M. Sartin. Combining content-based and collaborative filters in an online newspaper. In ACM SIGIR'99. Workshop on Recommender Systems: Algorithms and Evaluation, August 1999
[5] Pazzani, M. A framework for collaborative, content-based and demographic filtering. Artificial Intelligence Review, pages 393-408, December 1999.
[6] Billsus, D. and M. Pazzani. User modeling for adaptive news access. User Modeling and User-Adapted Interaction, 10(2-3):147-180, 2000.
[7] Soboroff, I. and C. Nicholas. Combining content and collaboration in text filtering. In IJCAI'99 Workshop: Machine Learning for Information Filtering, August 1999.
[8] Basu, C., H. Hirsh, and W. Cohen. Recommendation as classification: Using social and content-based information in recommendation. In Recommender Systems. Papers from 1998 Workshop. Technical Report WS-98-08. AAAI Press, 1998.
[9] Popescul, A., L. H. Ungar, D. M. Pennock, and S. Lawrence. Probabilistic Models for Unified Collaborative and Content-Based Recommendation in Sparse-Data Environments. In Proc. of the 17th Conf. on Uncertainty in Artificial Intelligence, Seattle, WA, 2001.
[10] Schein, A. I., A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In Proc. of the 25th Annual Intl. ACM SIGIR Conf., 2002.
[11] Condliff, M., D. Lewis, D. Madigan, and C. Posse. Bayesian mixed-effects models for recommender systems. In ACM SIGIR'99 Workshop on Recommender Systems: Algorithms and Evaluation, August 1999.
[12] Balabanovic, M. and Y. Shoham. Fab: Content-based, collaborative recommendation. Communications of the ACM, 40(3):66-72, 1997.
[13] Melville, P., R. J. Mooney, and R. Nagarajan. Content-Boosted Collaborative Filtering for Improved Recommendations. In Proceedings of the Eighteenth National Conference on Artificial Intelligence, Edmonton, Canada, 2002.
[14] Pazzani, M. A framework for collaborative, content-based and demographic filtering. Artificial Intelligence Review, pages 393-408, December 1999.
[15] Salton, G. Automatic Text Processing. Addison-Wesley, 1989.
[16] Rich, E. User Modeling via Stereotypes. Cognitive Science, 3(4):329-354, 1979.
[17] Powell, M. J. D. Approximation Theory and Methods, Cambridge University Press, 1981.
[18] Armstrong, J. S. Principles of Forecasting – A Handbook for Researchers and Practitioners, Kluwer Academic Publishers, 2001.
[19] O. Dekel, C.D. Manning, Y. Singer. Log-linear models for label ranking. Advances in Neural Information Processing Systems (NIPS-03), ed. by S. Thrun, L.K. Saul, B. Schölkopf (MIT, Cambridge, MA, 2004), pp. 497-504
[20] J. Fürnkranz, E. Hüllermeier. Pairwise preference learning and ranking. Proceedings of the 14th European Conference on Machine Learning (ECML-03), vol 2837. Lecture Notes in Artifical Intelligence, ed. by N. Lavrac, D. Gamberger, H. Blockeel, L. Todorovski (Springer, Cavtat, Croatia, 2003), pp. 145-156.
[21] S. Har-Peled, D. Roth, D. Zimak, Constraint classification: A new approach to multiclass classification. Proceedings of the 13th International Conference on Algorithmic Learning Theory (ALT-02), ed. by N. Cesa-Bianchi, M. Numao, R. Reischuk (Springer, Lübeck, Germany, 2002), pp. 365-379
[22] P.B. Brazdil, C. Soares, J.P. da Costa. Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results. Mach. Learn 50(3), 251-277 (2003)
[23] S. Rajaram, S. Agarwal. Generalization bounds for k-partite ranking. Proceedings of the NIPS 2005 Workshop on Learning to Rank, ed. by S. Agarwal, C. Cortes, R. Herbrich, (Whistler, BC, Canada, 2005), pp. 23-28
[24] J. Fürnkranz, E. Hüllermeier, S. Vanderlooy. Binary decomposition methods for multipartite ranking. ECML/PKDD-09, vol. Part 1. Springer 2009. pp-359-374
[25] A.P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recognit. 30(7), 1145-1159 (1997).
[26] M. Gönen, G. Heller. Concordance probability and discriminatory power in proportional hazards regression. Biometrika 92(4), 965-970 (2005).
[27] T. Joachims. Optimizing search engines using clickthrough data. Proceedings of the 8th ACM SIGKDD International

Conference on Knowledge Discovery and Data Mining (KDD-02) (ACM, 2002), pp.133-142
[28] F. Radlinski, T. Joachims. Learning to rank from implicit feedback. Proceeding of the ACM Conference on Knowledge Discovery and Data Mining (KDD-05) (2005), pp. 239-248
[29] K. Järvelin, J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. ACM Trans. Inf. Syst. 20(4), 422-446 (2002)
[30] C.D. Manning, P. Raghavan, H. Schütze. Introduction to Information Retrieval (Cambridge, University Press, 2008)
[31] X. Geng, T.-Y. Liu, T. Qin, A. Arnold, H. Li, H.-Y. Shum. Query dependent ranking using k-nearest neighbor. SIGIR-08. ACM, Singapore, 2008. pp. 115-122
[32] W. Ni, Y. Huang, M. Xie. A query dependent approach to learning to rank from information retrival. WAIM-08. IEEE, Zhangjiajie, China, 2008. pp. 262-269
[33] Johnson, William B.; Lindenstrauss, Joram (1984). "Extensions of Lipschitz mappings into a Hilbert space". Conference in Modern Analysis and Probability (New Haven, Conn., 1982). Contemporary Mathematics 26. Providence, RI: American Mathematical Society. pp. 189–206.
[34] Dmitriy Fradkin and David Madigan. Experiments with random projections for machine learning.KDD ’03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 517-522. ACM Press, 2003
[35] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.
[36] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen, Thomas Seidl. MOA: Massive Online Analysis, a Framework for Stream Classification and Clustering. Journal of Machine Learning Research (JMLR) Workshop and Conference Proceedings. Volume 11: Workshop on Applications of Pattern Analysis (2010)
[37] M. Sahami, S. Dumais, D. Heckerman, E. Horvitz (1998). A Bayesian approach to filtering junk e-mail. AAAI’98 Workshop on Learning for Text Categorization
[38] P. Domingos, G. Hulten. Mining High-Speed Data Streams. ACM SIGKDD Conference, 2000.
[39] W.W. Cohen, R.E. Schapire, Y. Singer. Learning to order things. J. Artif. Intell. Res. 10, 243-270 (1999)
[40] D. Coppersmith, L. Fleischer, A. Rudra. Ordering by weighted number of wins gives a good ranking for weighted tournaments. SODA-06. 2006. pp. 776-782
[41] J. Fürnkranz, E. Hüllermeier. Preference Learning. Springer, 2010.
[42] G. Tesauro. Connectionist learning of expert preferences by comparison training. NIPS-88. Morgan Kaufmann, 1989. pp. 99-106
[43] Yi Fang, Luo Si. A Latent Pairwise Preference Learning Approach for Recommendation from Implicit Feedback. ACM, 2012.
[44] A. Birlitiu, T. Heskes. Expectaion propagation for rating players in sports competitions. PKDD-07. Springer, Warsaw, Poland, 2007. pp. 374-381
[45] D. Bouyssou. Ranking methods based on valued preference relation: A characterization of the net flow method. Eur. J. Oper. Res. 60(1), 61-67 (1992)
[46] O. Chapelle, Y. Chang and T-Y Liu. The Yahoo! Learning to Rank Challenge. http://learningtorankchallenge.yahoo.com, 2010.
[47] Martin Szummer, Emine Yilmaz. Semi-supervised Learning to Rank with Preference Regularization. ACM, 2011.
[48] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In Proc. UAI ’09, pages 452-461, 2009.
[49] Lukas Lerche, Dietmar Jannach. Using Graded Implicit Feedback for Bayesian Personalized Ranking. Proc. RecSys ’14, pages 353-356, 2014.