Predicting E-commerce Consumer Behaviour Using Sparse Session Data

Øyvind Myklatun
Thorstein Kaldahl Thorrud

Master of Science in Computer Science
Submission date: June 2015
Supervisor: Helge Langseth, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

This thesis research consumer behavior in an e-commerce domain by using a data set of sparse session data collected from an anonymous European e-commerce site. The goal is to predict whether a consumer session results in a purchase, and if so, which items are purchased. The data is supplied by the ACM Recommender System Challenge, which is a yearly challenge held by the ACM Recommender System Conference.

Classification is used for predicting whether or not a session made a purchase, as well as what items it bought. Several characteristics of the data are analysed in order to discover what separates a buy-session from the rest. In addition the interactions with items will be analysed to see what items a given buy-session is likely to purchase. The data is on a rather general format containing only a session ID, an ID of the item interacted with, a timestamp, and a category of the object - meaning the analysis can be applicable to other e-commerce sites and domains. Observations from the analysis are used for extracting features and to provide other valuable information for the classification. The following algorithms for classification are evaluated: Random Forest, Logistic Regression, Decision Tree, Bayesian Network and Naive Bayes.

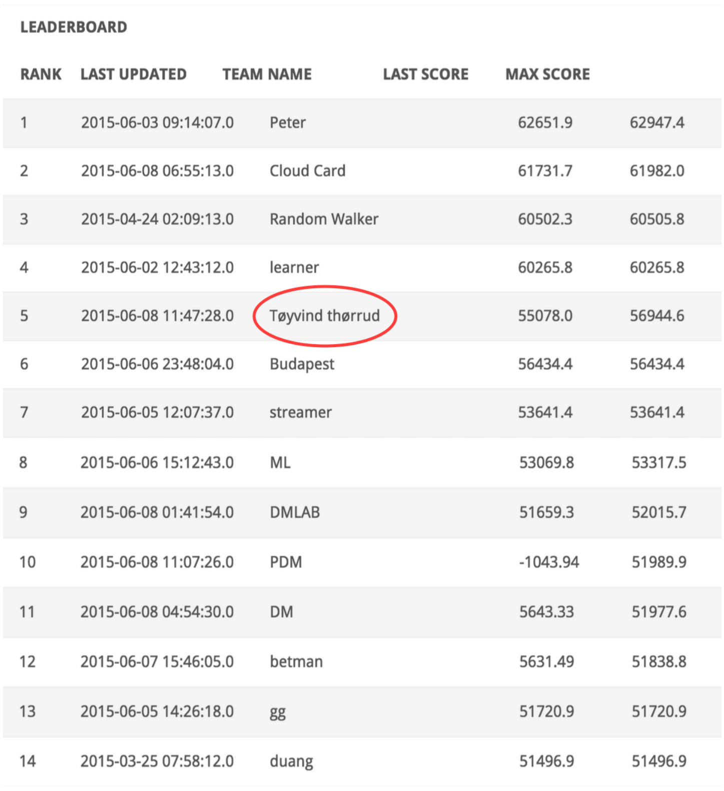
It is shown that one can predict a session's behaviour by using classification. Which items the session interacted with and when the interaction occurred proved to be important factors. The findings may contribute towards improving implicit ratings in recommender systems, or provide useful information for recommender systems when only session data is available.

Sammendrag

Denne oppgaven undersøker forbrukeroppførsel for e-handel ved å bruke et et datasett bestående av lite detaljerte forbruker-sessions. Målet er å predikere om en forbruker-session ender med å kjøpe noe, og hvis dette er tilfelle, hvilke produkter som kjøpes. Datasettet er levert av ACM Recommender System Challenge, som er en årlig konkurranse holdt av ACM Recommender System Conference.

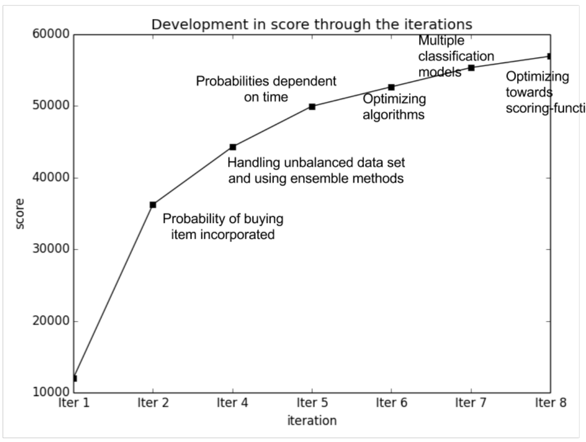
Det er brukt klassifisering for å predikere om en session kjøper noe eller ikke, og hvilke produkter som kjøpes. Flere karakteristikker ved dataen er analysert for å oppdage hva som skiller en kjøper fra en som ikke kjøper. I tillegg er interaksjonen en kjøper har med produktene analysert for å best mulig kunne predikere hvilke produkter han ender opp med å kjøpe. Dataen er heller generell og inneholder session ID, ID'en til produktet som er samhandlet med, et tidsstempel og en kategori for produktet. På denne måten kan analysen være nyttig for andre e-handelnettsider og domener. Informasjonen fra analysen blir brukt til å hente ut verdifulle attributter for klassifiseringsalgoritmene. For klassifisering har de følgende algoritmene blitt evaluert: Random Forest, Logistic Regression, Decision Tree, Bayesian Network og Naive Bayes.

Det blir vist at en kan predikere en session sin oppførsel ved å bruke klassifisering. Hvilke produkter og når interaksjonen foregikk viser seg å være viktige indikatorer. Funnene kan bli brukt til å forbedre nøyaktigheten til implisitte ratings i anbefalingssystemer, eller tilby nyttig informasjon for anbefalingssystemer hvor bare session data er tilgjenglig.



LEADERBOARD

RANK	LAST UPDATED	TEAM NAME	LAST SCORE	MAX SCORE
1	2015-06-03 09:14:07.0	Peter	62651.9	62947.4
2	2015-06-08 06:55:13.0	Cloud Card	61731.7	61982.0
3	2015-04-24 02:09:13.0	Random Walker	60502.3	60505.8
4	2015-06-02 12:43:12.0	learner	60265.8	60265.8
5	2015-06-08 11:47:28.0	Tøyvind thørrud	55078.0	56944.6
6	2015-06-06 23:48:04.0	Budapest	56434.4	56434.4
7	2015-06-05 12:07:37.0	streamer	53641.4	53641.4
8	2015-06-06 15:12:43.0	ML	53069.8	53317.5
9	2015-06-08 01:41:54.0	DMLAB	51659.3	52015.7
10	2015-06-08 11:07:26.0	PDM	-1043.94	51989.9
11	2015-06-08 04:54:30.0	DM	5643.33	51977.6
12	2015-06-07 15:46:05.0	betman	5631.49	51838.8
13	2015-06-05 14:26:18.0	gg	51720.9	51720.9
14	2015-03-25 07:58:12.0	duang	51496.9	51496.9



Preface

This project was conducted as a master thesis at the Norwegian University of Science and Technology (NTNU). The main supervisor was Helge Langseth, in addition to Anders Kofod-Petersen and Hai Thanh Nguyen at Telenor Research. We would like to thank the team for weekly meetings, where we shared ideas and had a lot of fun.

Thorstein Kaldahl Thorrud and Øyvind Herstad Myklatun
Trondheim, June 8, 2015

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter contains the background and motivation for this thesis, including a presentation of the ACM Recommender System Challenge 2015 (RecSys Challenge). The thesis goal and research objectives will be presented, in addition to our contributions to the research field of recommender systems. Finally, a short overview of the structure of the thesis is given.

1.1 Background and Motivation

Electronic commerce refers to the trading in products and services using computer networks, such as the Internet. Business to consumer e-commerce, or online shopping, is a form of e-commerce where a business targets individual consumers. Online shopping has grown steadily on a global basis the last years, and reached 1.2 trillion dollars in 2013. Globally 22 percent of all disposable income is spent online, and in the U.S online shopping provides around ten percent of the retail revenue [35].

Many of the largest online retail stores, such as Amazon, have implemented recommender systems to provide the consumer with a better shopping experience and to increase revenue. A recommender system is dealing with a set of users: $u_1, u_2, u_3 ... u_n$, and a set of items or products: $it_1, it_2, it_3, ..., it_m$. The number of items is often high and users have seldom visited all of them. Thus, there are only some of the items we can know the user's preference for. The user's preference can either be explicit or implicit. Only when the online store provides the users with the possibility of rating items it can have explicit information. The drawback with a system solely based on explicit information is that it depends on feedback from the users. Another way to get the user's preference is to analyze how the user behaves when visiting or watching a product. If a user visits the same product several times this should increase the user's rating of the product. Hybrid solutions, where implicit and explicit information are combined, are also possible.

Recommender systems try to predict ratings that a user would give to an unknown item, and in this way help the user in the process of both finding interesting items and deciding what items he should buy. In Table 1.1 it is given an example of a typical problem for a recommender system. The rows represent users and the columns represent products. The integers represent what rating a user has for the different items. As explained above, this information can either be explicitly given by the user or implicitly generated by the system. A recommender system's task is to predict the ratings missing in Table 1.1.

	D_1	D_2	D_3	D_4
U_1	-	-	4	5
U_2	4	-	3	-
U_3	3	5	4	4
U_4	-	5	-	5

Table 1.1: User-item matrix

There are mainly two different approaches when it comes to recommender systems: content based filtering (Lang [22], Van Meteren and Van Someren [39]) and collaborative filtering (Su and Khoshgoftaar [36]). Collaborative filtering methods predict a user's preference for an unknown item by looking at what preferences a group of other users has for the same item. One way of doing collaborative filtering is to look at what similar users have expressed interest for. Such users can be found by for example looking at the preferences two users have. If their preferences are adequately alike the two users are said to be similar. When using content based filtering one looks at what items a user has visited earlier when making predictions about other items the user may like. Each user in a content based recommender system is represented with a profile telling the user's preference. This profile is built by examining items that the user previously has watched. For a movie recommender a user profile could exist of actresses the user likes and the movie genres he has enjoyed watching. When the system is to recommend new items to a user it will try to find other items that matches the user profile of the given user.

1.1.1 RecSys Challenge

The ACM Recommender Systems Conference (RecSys) is the premier international forum for the presentation of new research results, systems and techniques in the broad field of recommender systems. Each year RecSys has a competition

[1]. This year's competition is to predict whether a session from an e-commerce store ends up buying, and if so, what items it bought. The self proclaimed goal of this contest is to get more information about what encourages a user to become a buyer and for e-business companies to be better able to suggest items to the consumers - thus improving the recommender systems. By participating in the competition we want to contribute towards this.

RecSys has handed out both training and test data. The training data consists of clicks performed by sessions, and a file containing session IDs and what these sessions ended up buying. The test data consists of clicks performed by sessions, and the task is to predict if these sessions bought something, and if so, what items they bought. The solution one produces can be uploaded to the RecSys Challenge home page and be evaluated with a score. On the home page one can also find a score board telling how good one's own solution is compared to others.

A session can buy several items, and also more than one of an item. The task excludes to predict how many of an item a session buys, only if the session buys the item or not. When one has predicted which sessions that bought and what items these sessions bought, a score is computed as follows:

$$Score(\mathbf{Sl}) = \sum_{\forall s \in \mathbf{Sl}} \begin{cases} \frac{|S_b|}{|S|} + \frac{|A_s \cap B_s|}{|A_s \cup B_s|} & \text{if s} \in S_b \\ -\frac{|S_b|}{|S|} & \text{else} \end{cases} \tag{1.1}$$

where **Sl** is the sessions you have predicted buy for; S_b is the sessions that actually have bought something; S is all the sessions; A_s is the set of items that a session actually have bought; B_s is set of items that one has predicted the session to buy. $\frac{|S_b|}{|S|}$ is the share of sessions buying something. As one see in Section 3.1 this approximates to 0.055. The maximum score one can get out of one session is 1.055 and is obtained by guessing all the items a session bought correctly, without guessing any wrong items.

The information handed out is rather sparse. There is no information about the company that has supplied the contest with the data, only that it is a European e-commerce company selling everything from garden tools to electronics. The data used in the contest is also sparse. There is no information about what user a session belongs to, thus every session must be considered as a separate user. Further, the only type of event contained in the data is items clicked by a session. There is no information about the session browsing, searching, adding to chart or so on. In the clicks performed a session there is given information about: what item ID the item has; what category it belongs to; and when the item was clicked. If an item ended up being bought by a session, we have information about: when this happened; the price of the item; and the quantity the session bought of this item. More information about the data can be found in Section 3.1.

1.1.2 Approach

The challenge will be looked at as containing two tasks. The first task is to decide whether a session ended up buying or not. This clearly stands out as a binary classification task, classifying sessions as *buy* or *not-buy*. The other task will be to decide which of the items a predicted buy-session visited actually ended up being bought. This will also be considered as a classification task. The two tasks will be referred to as buy-or-not classification and items-bought classification. The main focus of this thesis will be to analyse the data at hand and to extract valuable features and other information which can be used for deciding if a session ends up buying, and in that case, what it bought. From the process of analyzing the data, extracting information from the analysis and predicting the behaviour of a session, triggering factors for what affects a consumer to buy and what it buys will be presented.

The process for deriving at the final solution is conducted in an iterative way. The first step in this iterative process is to analyse the data. This information is then used for evaluating how the solution can be improved. Improvements are done by: extracting new features; improving features; evaluating different classification algorithms; parametrization of the algorithms; and/or optimizing our solution towards the RecSys Challenge scoring-function.

1.2 Goals and Research Questions

Goal Predict buys of online shoppers based on sparse session data using classification

Given a set of clicks in a session from an online store, it will be predicted whether the session ended with a buy or not. Given that the session was predicted a buy it will also be predicted what items the session bought. The only information available is: the session ID, the item ID of the item clicked; a timestamp of when the click occurred; and the category and price of the item. To solve this problem we have decided to split the process in two sub problems. These two sub problems are presented in the research objectives:

Research objective 1 Decide whether a session ended up buying or not.

Research objective 2 Given that a session was predicted as a buy, decide which items that were bought.

If we can accomplish these objectives we will be able to provide information that will be useful for the broad research field of recommender systems.

1.3 Contributions

We have shown that one can gather information about an online e-commerce user's preference solely based on anonymous sparse session data. This information has been used for predicting if a session ended up buying, and it that case, what items it bought. Given the concerns regarding privacy and leakage of information, recommender systems face a challenge in the future of being able to cope with anonymous data. The information provided in this thesis may be valuable for such situations, as well as when making implicit ratings that are to be used in a recommender system.

1.4 Thesis Structure

In Chapter 2 the terminology and concepts needed to understand the master thesis is explained. Recommender systems are described in greater detail, and some approaches for building one is shown. Theory related to the task of classification will be presented. This includes feature extraction, feature subset selection, algorithms and evaluation measures.

The data handed out from the RecSys Challenge will be described in greater detail in Chapter 3. It will be discussed how the data was pre-processed to get as much information from it as possible. Further, different aspects related to a session ending up buying and what it buys will be analysed. The analysis is meant to be rather general, so it can be applied to other online stores having the same information available.

After the data analysis, the iterative process of finding a solution for the RecSys Challenge is presented. Although the data analysis and the iterative process are two different chapters, the analysis of the data is conducted as a part of the iterations. The results from the iterations are evaluated in Chapter 5. Further, it is discussed how the results can be useful for recommender systems.

Chapter 2

Background Theory

2.1 Introduction

It will be introduced terminology and theory needed to understand the rest of the master thesis. As explained in the introduction the task is to predict whether a session from an online shopping site ends up buying or not, in addition to predicting the items bought. Both these problems will be considered as classification tasks. This chapter will explain the task of classification, important terminology related to this and introduce the classification algorithms used. Moreover, concepts of how one can get the best performance out of the classification algorithms are presented. Theory about recommender systems will also be presented as this is one of the fields of interest of this master thesis.

2.1.1 Recommender systems

Recommender systems (RS), surveyed by Bobadilla et al. [3], collect data of user's preferences in a long range of domains such as: books, movies, songs, gadgets, travels, websites and so on. This information is gathered either explicitly, by for example allowing the users to give explicit feedback to a given service or item, or implicitly by monitoring the user's behavior. The goal of collecting all this data is then to give the users recommendations of what they might be interested in based on their explicit or implicit feedback to the system.

The long tail

In order to understand one of the advantages with recommender systems it is necessary to look at a concept called long tail. In the physical world, it is not possible to tailor a store to each individual consumer. Thus the solution is simply to display the most popular products, which is the solution that best fits the total consumer base.

However as commerce has moved on to use the world wide web as a marketplace, this solution is not necessarily the optimal one. The advantage of e-commerce is that the retailer is able to sell all available products to the user, as there are no physical restrictions in shelf space. The concept of long tail describes the popularity of items, where there is a small percent of items that has a very high popularity among the users compared to the majority of items which thus forms a long tail. For e-commerce it will be desirable to tailor the displayed products to each individual users' preferences. The first reason is that it is not possible to present all the available items, as this often is a vast amount, and the second is that one can not expect the users to have heard of all items they may be interested in.

Methods

As mentioned in Section 1.1, one of the approaches for creating recommender systems is collaborative filtering. This can be done by using *latent factor models* (Takács et al. [37]). Latent factor models predict preferences by characterizing both users and items with a set of factors inferred from the existing preferences. This is done by using matrix factorization (Koren et al. [20]). Initially one has a matrix, $\mathbf{R}$, where the columns are representing items, D, and the rows are representing users, U. From each row we can read the preference the user has for each item, see Table 2.1. One wants to find two matrices $\mathbf{P}$ and $\mathbf{Q}$ such that $\mathbf{P} \cdot \mathbf{Q} = \hat{\mathbf{R}} \approx \mathbf{R}$, where $\mathbf{P}$ is a $|U| \cdot k$ matrix and $\mathbf{Q}$ is a $k \cdot |D|$ matrix and k is an integer larger than 1 representing the number of latent factors to use. In this way, $\mathbf{P}$ represents the users and their association with the latent factors and $\mathbf{D}$ represents the items and their association with the latent factors. Making $\hat{\mathbf{R}}$ as close to $\mathbf{R}$ as possible means minimizing the difference to the values in $\mathbf{R}$. When trying to make $\hat{\mathbf{R}}$ as similar to $\mathbf{R}$ as possible one starts out assigning random values to the matrices $\mathbf{P}$ and $\mathbf{Q}$ and iteratively tries to minimize the difference between the product and $\mathbf{R}$ by using for example gradient descent (Takács et al. [37]). The result is a matrix, $\hat{\mathbf{R}}$, that has ratings for all items for all users.

	D_1	D_2	D_3	D_4
U_1	-	-	4	5
U_2	4	-	3	-
U_3	3	5	4	4
U_4	-	5	-	5

Table 2.1: User-item matrix

Other ways of doing collaborative filtering are to apply an item-oriented or a user-oriented approach (Koren et al. [20]). With the item-oriented approach one predicts the preference of an unknown item for a user by looking at the ratings of *neighbouring items* by the same user. An item's *neighbouring items* are items that tend to get similar ratings when rated by the same user. A popular user-oriented approach is based on finding similar users by using kNN(k nearest neighbours) [4]. For a user a, kNN is used for finding the neighborhood of a. Then, all ratings for unrated items by a is collected from the users in a's neighborhood. Finally, the scores from the neighborhood are aggregated and result in an ordering of what items the system should recommend to a.

Content based filtering is tightly knitted to the field of information retrieval, and many of the content based recommender systems use relatively simple retrieval models using keyword matching or the Vector Space Model(VSM) to find items that matches a user's profile (Lops et al. [24]). In the VSM each user and item is represented as a n-dimensional vector where each dimension represents a term from the overall vocabulary. The vectors consist of term weights telling how related an item or user is to the given term. When using the VSM one has to decide how these term weights should be assigned, in addition to how one should measure the closeness between a user-vector and an item-vector. The items the system recommends to the user are the items that are the most similar to the user-profile given the similarity measure.

One of the challenges with recommender systems is the *cold start problem.* When a user has not interacted sufficiently with the system the system fails to deliver good enough recommendations. This can happen when new users arrive to the system, or when users do not use the system enough.

2.2 The task of classification

The task of classification is about classifying an object as one of the pre-specified set of classes. For example to determine what kind of species a certain animal belongs to. To do this one has to find out what distinguishes a class from the others. If one are to determine whether an animal is a zebra, one have to look at the animal and see if the animal fits the *core* description of a zebra. One of the biggest characteristics with zebras are the stripes, so it would be nearby to look at that before looking at for example the ears of the animal. After one has looked at the animal and can confirm that it has stripes one can continue the process by selecting another characteristic, after all there are also other animals that have stripes, like tigers.

In the field of machine learning the process of classifying as explained above, is often referred to as supervised learning. In supervised learning one knows the class and characteristics for a set of samples, called training data, and the goal is to make a function that takes characteristics as input and outputs a class. Formally one want to make a function f that takes a sample $\mathbf{x}$ as input. The sample $\mathbf{x}$ is a vector where each dimension represents a certain property. Based on the properties of $\mathbf{x}$ the function outputs a class, often together with a probability of that sample belonging to that class. In this way one can use the function to predict what class an unknown sample belongs to. The quality of the function heavily depends on the quality of the training data and the differences between the distinct classes. If the differences between the classes are small it is obviously harder to make a function that outputs correct class for a given sample. The function made from the training data is referred to as a model.

2.2.1 Data representation

When dealing with supervised machine learning tasks the data is often represented in a table. Each row in the table represents an *example*, *instance* or *sample*, where the notation *sample* will be used in this thesis. Each sample is represented with a set of attributes or features - where feature will be the notation used - together with a class label. An example of such a representation can be seen in Table 2.2. Displayed is a sample representing a click from an e-commerce session with *Session ID*, *Item ID*, *duration*, *clicks* and *first*. The features are typically nominal or numeric. A nominal feature is a feature where the samples can take values from an unordered set, while numeric features are real numbers. An example of a nominal feature is whether an item was the first item clicked in an online shopping session. *True* is represented by 1 and *false* by 0. An example of a numeric feature is the duration of the session, which can take any real number above 0.

Supervised machine learning optimally requires separate training and test sets. The training set is used for building a model that can predict the class of unknown samples. The test set is is used for testing the model built from the training data. When testing with the test data the class labels of the test-samples is not given. The ability of the model to correctly predict the class labels of the test-samples tells us the performance of the model.

SessionID	ItemID	duration	clicks	first	class
11	3743	783,761	12	1	buy

Table 2.2: Sample format

2.2.2 Feature extraction

Often it is possible to capture the information in a data set much more effectively by creating a new set of features from the original features. Moreover, it may be possible to reduce the number of features you need. Feature extraction is such a process, and involves creation of new features from the raw data. Consider clicks from a session on an online store containing only the item clicked and the timestamp. You are to decide whether the session ended up buying something or not. The timestamp itself could be interesting, but also opens a lot of other possibilities. For example you can extract the duration of the session; the weekday; and the total time the session spent watching an item.

Feature construction can also be seen as part a of the process of extracting features from the data set. Some type of features are incompatible with some algorithms. An example are certain decision trees that need to have the features on a categorical form. Thus, it is often necessary to modify continuous features into categorical features - called discretization. An example can be the duration in the sample above. The duration can be everything from 0 seconds to many hours, but some algorithms only takes features that are categorical. One could then split the duration feature into three categories: 0-100 seconds; 100 - 500 seconds; and 500 seconds and above.

2.2.3 Discretization

There are several methods for discretizing continuous values, however the only one that will be presented is the method by Fayyad and Irani [13], as this is the one used by the machine learning software Weka (see Section 2.5). To discretize a continuous feature means to split the possible values of the feature into categories, thus obtaining a finite number of possible categories, making it possible to treat the feature as a discrete one.

The first step of the approach is to sort the samples of the continuous feature one wish to discretize, so that they are in increasing order for the feature's values. Each point, where a point is between two samples, is then evaluated as potential cut points. Let the cut point partition the set of samples S into sub sets S_1 and S_2; where there are k classes $C_1, ..., C_k$; and $P(C_i, S)$ is the proportion of samples in S belonging to class C_i. To evaluate a cut point, the method uses a formula for calculating the entropy of each category created by the cut:

$$Ent(S) = -\sum_{i=1}^{k} P(C_i, S)log(P(C_i, S)) \tag{2.1}$$

To compare two resulting sets S_1 and S_2 after a partition for feature A and with the cut value T, the following equation is evaluated:

$$E(A,T:S)=\frac{|S_1|}{|S|}Ent(S_1)+\frac{|S_2|}{|S|}Ent(S_2) \tag{2.2}$$

This measure is applied recursively to the two sets from the previous split, always selecting the split that minimizes the measure in equation 2.2, until some stopping criteria is achieved. The stopping measure used by Fayyad and Irani [13] is based on a principle called The Minimum Description Length Principle. The minimum description length of an object is the minimum number of bits required to describe that object uniquely. Furthermore the principle states that a hypothesis HT, where length added with length given the hypothesis is: $MLength(HT)+MLength(S|HT)$, should be chosen if this value is less than all other hypotheses. The length in this case is measured in bits. Fayyad and Irani [13] arrived on the following equation for stopping criterion:

$$Gain(A,T;S)>\frac{log_2 N-1}{N}+\frac{\Delta(A,T;S)}{N} \tag{2.3}$$

Thus a partition by cut point T is accepted if equation 2.3 is satisfied. Dougherty et al. [12] tested several discretization algorithms with decision trees and Bayesian classifiers. A discretization technique similar to the one used by Fayyad and Irani [13] performed slightly better than the others.

2.2.4 Feature subset selection

It can be tempting to think that more features mean more information, and choosing a subset of the features would only cause loss of information. This is not necessarily the case if the feature is irrelevant or two features correlate (Blum and Langley [2]). Irrelevant features are features which provides little or no information at all. An example can be the session ID of an on-line shopper when one is to decide whether the session bought something or not. Redundant features are two features giving the same information, they are highly correlated with each other. For example one could imagine that the number of clicks a session has strongly correlates with the duration of the session. Such features can create a bias in the classification models, which again can affect their performance.

Another reason why it could be beneficial to perform feature subset selection is to avoid the *Curse of Dimensionality*. This phenomenon describes the problem arising when the feature value space is vast, which requires a enormous number of training samples in order to ensure that there are several samples for each combination of feature values. For classification this problem arises when you have too few samples to allow the creation of a model that reliably assigns a class to all samples.

Kohavi and John [19] developed a method for feature subset selection, referred to as a wrapper, which considers how a specific classification algorithm interacts with a particular training set. The approach for solving this consists of applying a search method for finding a good subset, and using the classifier itself as part of the evaluation measure for the search result. There are several options of what search algorithm to apply, of which greedy search, also called hill-climbing, will be presented here. The greedy search algorithm is rather trivial: by first choosing an evaluation measure, such as accuracy or ROC (see Section 2.4), the single feature performing best is first selected; then it measures the result by adding a new feature, and selects the one giving the best result together with the first one. The search ends when no features improve the evaluation measure more than a certain threshold. This wrapper method is implemented in Weka (see Section 2.5).

2.2.5 Unbalanced data set

A data set is said to be unbalanced when the classes are not approximately equally represented Chawla [9]. An example of such a data set is the data set in the RecSys Challenge. The data consists of sessions from an online store that can take two different class values, *yes* and *no*. *Yes* means that the session ended up buying something, and *no* means that no products where bought. For every buy-session there are approximately 17 sessions that do not buy. The class that is represented with the least samples are called the minority class (*yes*), while other classes make up the majority(*no*). Most classification algorithms pursue to minimize the percentage of incorrect predictions of class labels Chawla [9]. When dealing with an unbalanced data set this is often not desirable. For the data set from the RecSys Challenge a classifier could have labeled all samples as no and got only approximately 5 percent misclassified samples. Moreover, when dealing with unbalanced data sets it is often the minority class that is interesting.

Several approaches for handling unbalanced data sets have been suggested in the literature. The three most common are different forms of *undersampling*, *oversampling* and *cost-sensitive learning*. Given an unbalanced data set, undersampling methods remove samples from the majority class until the classes are more equally represented (Kubat and Matwin [21], Laurikkala [23]). When undersampling you suffer the risk of removing important information in the majority class. Oversampling, opposed to undersampling, makes use of the minority class to construct more samples. The simplest approach when doing oversampling is to randomly select positive samples, and add them to your data set once more. In this manner you are using the information you already have, and suffer the risk of overfitting (Chawla [9]). The SMOOTE algorithm was introduced by Chawla et al. [8] to mitigate this problem. The algorithm uses the feature space instead of the sample space when performing oversampling. For each positive sample they construct a synthetic sample by merging the features of the k nearest neighbours.

When using oversampling or undersampling one modifies the data set at hand. Cost-sensitive learning mitigate this by assigning different costs for classifying a sample to a given class. If one wants more positive samples classified as positive one can assign a higher cost for classifying a positive sample as a negative. This can either be incorporated into the classifier model building process, or by using different sampling techniques. Not all algorithms have the ability to incorporate cost-sensitive learning into their building process (Weiss et al. [40]).

Weiss et al. [40] conclude that for unbalanced data sets it can not be drawn conclusions towards what is the better approach. Further, they say that for small data sets oversampling outperforms undersampling.

2.2.6 Overfitting/Underfitting

When using supervised learning for classification, a common phenomenon is overfitting (Dietterich [11]). Overfitting occurs when building a model on a training set which fits the data too well, in that it fits the noise and peculiarities that is not representative of the domain, but only in the training set. This is why the optimization problem of mapping the training set features to the training set classes should not be formulated as "minimize error on the training data". Several methods has been developed for coping with this problem, such as regularization methods, minimum-description-length methods and generalized cross-validation. A simple way of avoiding this is to measure the error of the model on a separate test set, containing only samples that is not used to build the model.

The concept of underfitting, is at the other end of the scale, and describes a model that is too regularized so that the model suffer from information loss. Thus the goal when building a model, is to capture all general information from the training set, but to avoid modelling the noisy part of the data.

2.3 Classification Algorithms

One focus of this thesis is to extract useful features from sessions in an e-commerce site, in order to classify whether or not these sessions will buy, and if they do, what they will buy. Classification problems are a typical machine learning problem, and can be generalized to apply in several domains.

In order to classify sessions, there will be used software named Weka and scikit-learn (see Section 2.5). This section will describe the theory behind the algorithms used in order to understand how the data is classified.

2.3.1 Bayesian Networks

A Bayesian Network is a directed acyclic graph, where nodes represents a random variable and contains its probability information (Friedman et al. [15]). When a node A is pointing to a node B, it is said that A is the parent of B, where A can have several parents. Each node has a conditional probability distribution $P(X|Parents(X))$, which means that the probability of X is decided as a combination of all its parents. The Bayesian Network has an important feature in that neighboring nodes, meaning nodes with common parents, are conditionally independent. Conditional independence means that two variables are independent given a third variable which is known.

To find the joint probability distribution from a Bayesian network, one applies the following equation:

$$P(x_1, ..., x_n) = \prod_{i=1}^{n} P(x_i|parents(X_i)) \tag{2.4}$$

In order to comprehend the use of a Bayesian network for classification problems, there will be presented a simplified example of how to calculate the probability of an e-commerce session being a buy session. The example makes use of two features: *clicks*, *duration*; and a class variable *buy* being *yes* or *no*. The feature *clicks* is a discrete feature and *duration* is a continuous. Lets say one wants to find the probability of *buy* being *yes*, *clicks* being 3, and *duration* being less than 20 seconds. Let the conditional probability tables in the network state that: $P(clicks = 3) = 0.6$, $P(duration < 20) = 0.5$, $P(class = YES)|clicks = 3, duration < 20) = 0.2$. The joint probability will then be calculated as follows:

$$\begin{aligned} &P(class = YES, clicks = 3, duration < 20)\\ &= P(class = YES|clicks = 3, duration < 20) \cdot P(clicks = 3) \cdot P(duration < 20)\\ &= 0.2 \cdot 0.6 \cdot 0.5 = 0.06. \end{aligned} \tag{2.5}$$

Notice that the feature *duration* is a continuous value, and thus needs to be discretized in order to work as input to the Bayesian network. The approach described in Section 2.2.2 can be applied to do this.

The task of learning a Bayesian Network can be stated as follows: Given a set of training samples D, find a network B that best matches D. To solve this task, a scoring function is applied in order to evaluate the different possible networks, a common one being the minimal description length (MDL) scoring function. The MDL principle is concerned with finding a model that provides the shortest description of the original data, which incorporates both the description of the model and the data using the model. By using a encoding scheme to describe the model representing the probability distribution over the data, that assigns shorter code words to more probable instances, the goal is then to minimize the combined description length of the network description and the encoded data. Let B be a Bayesian Network; $D = u_1, ..., u_N$ is the training set where u_i set the value to all features in the set; and $|B|$ is the total number of possible values for all nodes in the network, except those containing a class label; the MDL function is given by:

$$MDL(B|D) = \frac{\log N}{2}|B| - LL(B|D), \tag{2.6}$$

where $LL(B|D)$ is the log likelihood of B given D:

$$LL(B|D) = \sum_{i=1}^{N} log(P_B(u_i)) \tag{2.7}$$

The higher the log likelihood value, the closer the network B is to model the probability distribution of the data D. However it is not sufficient to only apply the log likelihood function in order to measure the quality of a network, as it tends to favor a highly connected, complete network. By applying the MDL where the number of nodes and their possible values are taken into account, one is able to avoid overfitting of the training data.

2.3.2 Naive Bayes

The Naive Bayes classifier, studied in Rish [31], is a rather simple classifier that assumes conditional independence between features given the class of a sample. Even though this generally is a poor assumption to make, Naive Bayes turns out to compete well to other classifiers in practice.

Given a set of features $X = (X_1, ..., X_n)$; and a set of training data D with samples $D_i = x_1, ..., x_n$, where x_i is the value for feature X_i; and a class label C; the Bayes rule can be applied to find the probability $P(C_i|D_i)$. In order to find the apriori probabilities, that is the probabilities calculated from the training data, Bayes rule is applied as follows:

$$P(C = i|\mathbf{X}{=}\mathbf{x}) = \frac{P(\mathbf{X}{=}\mathbf{x}|C = i)P(C = i)}{P(\mathbf{X}{=}\mathbf{x})} \tag{2.8}$$

The bold $\mathbf{X}{=}\mathbf{x}$ symbolizes that this is the feature vector $\mathbf{X}$ having a value vector $\mathbf{x}$. Further, as $P(\mathbf{X}{=}\mathbf{x})$ is the same for all classes this can be ignored. Since a large feature space would yield endless of combinations that need a probability, the assumption of independence is made. By applying this assumption, the classifier can be made up by calculating the probability of each possible value of each feature given the class, based on the training data. When a new sample is to be classified, the maximum probability of the following equation decides which class the sample belongs to:

$$f(\mathbf{x}) = \prod_{j=1}^{n} P(X_j = x_j|C = i)P(C = i) \tag{2.9}$$

As stated, the class $C = i$ that gives the highest probability will be the predicted class for the observed sample $\mathbf{x}$.

2.3.3 Decision Tree

Several have researched the concept of decision trees as classifiers (Breiman et al. [6]), and a survey were done by Safavian and Landgrebe [32]. Decision trees are a widely used method for classification, with a fairly intuitive structure. The structure is hierarchical and consists of nodes and directed edges, with three different types of nodes: the root node has no incoming edges and zero or more outgoing edges; the internal nodes have exactly one incoming edge and two or more outgoing edges; and the leaf nodes have exactly one incoming edge and no outgoing edges. The root node and the internal nodes contain a test condition that split the samples being fed into the tree on some feature. When the sample has been parsed through the decision tree, it ends up on a leaf node which contains a class label assigned to this sample. To start classifying samples, one first need to build a decision tree on some training set. The theory of this section is based on Hunt's algorithm, which uses a greedy strategy for building a tree.

Let D_t be a set of training samples associated with node t and let $y = y_1, y_2, ..., y_n$ be the class labels. A decision tree is then built by two recursive steps:

1. If all the samples in D_t has the same class y_t (or some other criteria is met), then t is a leaf node with the class label y_t. An example of another criteria is if the number of samples in this node is below some pre-defined threshold. This criteria is often set in order to reduce the risk of overfitting.

2. If D_t contains samples with different class labels, a feature is chosen as a test condition to split the records into smaller subsets. For each outcome of the test condition, a child node is created and the samples in D_t are distributed among the child nodes based on their value of the feature used as test condition.

This procedure raises two issues. A child node created by step two above might be empty if there are no samples in the training set with the combination of feature values needed to end up at that child node. The solution to this problem is to make this node a leaf node with class label equal to the majority of samples in the parent node. The second issue is the possibility that the leaf node created contains samples with different class values while having the same feature values. Here the solution is to assign the class label of the majority samples in the set for this node.

The final consideration to be made when creating a decision tree is how to choose the feature to use as test condition to split the data. The tree is built in a top-down manner where for each node, with the given sample set residing on this node, each feature is considered in terms of how much information can be gained by splitting on each feature. Section 2.4.1 contains the evaluation measures on how to compare features relevant for this thesis.

2.3.4 Random Forest

Breiman [5] developed the idea of decision trees further by applying several trees instead of just one, which makes up the classifier Random Forest. The concept in simple terms is to create several independent trees, and for each sample to be classified, every tree casts a vote on which class it should be classified as.

One of the methods used in Random Forests by Breiman [5] is Bootstrap Aggregating, often called bagging. This method randomly chooses a subset of about two-thirds of the training set with replacement to build a single tree. As described in the paper, this is done as it seems to enhance the accuracy when used together with Random Feature Selection (RFS). RFS is another method used for building Random Forests, which consists of randomly choosing a fixed number of features to consider for each split in each tree. The concept of selecting random features for each split is used as it has proved to give lower generalization errors in that the trees are more robust to outliers and noise.

2.3.5 Logistic Regression

In order to understand the use of Logistic Regression, it is necessary to first look at the concept of Linear Regression (Seal [34]). Given several data points with input x and output y, the goal of Linear Regression is to find the line h_w that best fits all the points. The line will have the equation $h_w(x) = w_0 + w_1 \cdot x$, where w_0 and w_1 are the weights we want to learn. In order to find the best line, the empirical loss has to be minimized such that as little information as possible is lost. One way of doing this is to use the squared loss function L_2:

$$Loss(h_w) = \sum_{j=1}^{N} (y_j - (w_0 + w_1 \cdot x_j))^2 \tag{2.10}$$

Alternatively it is also possible to use the L_1 loss function, which minimizes the sum of absolute values instead of the sum of squares. Using L_1 regularization tends to produce a more sparse model, which can be less likely to overfit. The loss function is minimized when its partially derivatives with respect to w_0 and w_1 are zero. When moving on beyond linear models, minimizing the loss function can be done with gradient descent instead of partial derivation. The data points may also have more than one variable, which is often the case in classification. With multiple variables, a sample x_j is an n-element vector and the regression line would be on the form:

$$h_w(x_j) = w_0 + w_1 \cdot x_{j,1} + ... + w_n \cdot x_{j,n} = w_0 + \sum_i w_i \cdot x_{j,i} \tag{2.11}$$

Since the weight w_0 stands out from the other weights, a dummy input feature $x_{j,0}$ that is always equal to 1 can be used. The function will then simply be the dot product of the weights and input vector:

$$h_w(x_j) = w \cdot x_j = \sum_i w_i x_{j,i} \tag{2.12}$$

By performing gradient descent the squared-error loss can be minimized. Linear Regression can then be used to make a linear classifier by building a line on a set of training samples, and then feed in samples to be classified which then will be classified dependent on whether or not they are over or under the regression line. The linear classifier will either classify a sample as 0 or 1, even if the sample is very close to the boundary line. A sample of two variables will be multiplied with the generated weights like the following:

$$c(x) = w_0 + w_1 x_1 + w_2 x_2 \tag{2.13}$$

Where the sample is classified as 0 if $c(x) < 0$ and as 1 if $c(x) \geq 1$. This can be a problem as we often want to know how certain the classifier is for each classification, as there are many domains that require high probability of being correct before accepting the classification.

Cox [10] developed Logistic Regression, which uses a logistic function to model the classifier. This function outputs a real number between 0 and 1 for a sample and can thus be interpreted as the probability of the sample belonging to the positive class. Other than that, the samples and weights are on the same format. The logistic function is presented in the following equation:

$$h_w(x) = Logistic(w \cdot x) = \frac{1}{1 + e^{-w \cdot x}} \tag{2.14}$$

An input at the center of the boundary region will yield a probability of 0.5. Logistic Regression is the process of fitting the weights of this model so that the information loss on a data set is minimal. The standard way of minimizing the derivative of the squared-loss function L_2 is gradient descent for Logistic Regression as well as Linear Regression. However in practice, algorithms are often implemented with variants of optimization methods, as some methods perform better on data of specific characteristics.

2.4 Evaluation Measures

In order to evaluate which algorithms and features perform best, it is necessary to apply evaluation measures. There will be introduced evaluation measures for both features and classification algorithms.

2.4.1 Feature Evaluation

Related to what features that contributes the most when solving a classification task, two evaluation measures are presented. These are information gain and information gain ratio, which are calculated using an impurity measure.

Impurity Measures

In this section, two impurity measures will be considered for measuring the impurity of a set of samples given their classes. The measures are entropy and Gini. Let $p(i|t)$ represent the fraction of samples that belongs to class i at the node t, and c is the number of possible classes. The equation used to measure the entropy is:

$$Entropy(t) = -\sum_{i=0}^{c-1} p(i|t) log_2 p(i|t) \tag{2.15}$$

The gini impurity measure does not use the logarithmic function, but the square instead. Following is the equation for calculating gini:

$$Gini(t) = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2 \tag{2.16}$$

Information Gain

In order to know how much information a feature can give, with regard to the classes, it is necessary to compare the impurity before splitting the samples on the given feature with the impurity after splitting. As an example, lets say an animal is going to be classified as a land animal or an ocean animal. If the samples have a feature named *has_legs*, and for simplicity all land animals have legs and none of the ocean animals have legs. Then the information after splitting on this node would increase 100 percent as all the samples of class land animal would end up in node 1, and all the samples of ocean animal would end up in node 2.

The goal is to achieve highest possible information gain by splitting a given set of samples on a feature. Let $I(.)$ be the impurity measure of a given node; $N(v_j)$ be the number of samples belonging to child node v_j; N be the number of samples at the parent node; and k be the number of feature values. The information gain by performing the split is then:

$$InfoGain = I(parent) - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j) \tag{2.17}$$

Thus a lower impurity measure for the child nodes will yield a higher information gain value. The resulting split is also weighted by the number of samples residing in each node. If one child node has a low impurity, but also a low number of samples, this will not contribute as much to the information gain as if the child node had many samples.

Gain Ratio

Equation 2.17 will favor the split with lowest impurity, which is desirable but only to a certain degree. Imagine if the node splits samples based on a feature containing an unique value for each sample, such as an user ID. This kind of split will be useless as it will not create a generalized tree that can be used in practice. In order to detect such splits, a measure called gain ratio is applied instead of the information gain. This is done by first calculating the split info in the following way, where k is the total number of splits and $P(v_i)$ is the portion of samples being assigned to node v_i:

$$Split\ info = -\sum_{i=1} kP(v_i)log_2P(v_i) \tag{2.18}$$

Let IG be the information gain and SI be the split info, the gain ratio is then calculated:

$$Gain\ ratio = \frac{IG}{SI} \tag{2.19}$$

The split info will be higher the more child nodes that are needed, and the gain ratio will thus be lower. Gain ratio gives a measure of how much information gain a feature can give, by also taking into account how generalized this information can be used.

2.4.2 Algorithm Evaluation

Different metrics for evaluating classification algorithms will be presented. Their ability to handle unbalanced data sets will also be considered, as the data set from the RecSys Challenge is highly unbalanced.

Precision and Recall

Precision and recall are two widely employed metrics in binary classification problems with unbalanced data sets where the minority class is considered more interesting than the majority class (Tan et al. [38]). A sample from the majority class is referred to as a negative sample, while a sample from the minority class is referred to as a positive sample. In Table 2.3 a confusion matrix that summarizes the number of instances predicted correctly or incorrectly for a binary classification problem is shown. Precision an recall is defined as follows

$$P = \frac{TP}{TP + FP} \tag{2.20}$$

$$R = \frac{TP}{TP + FN} \tag{2.21}$$

Precision represents the fraction of samples that the classifier has predicted as positive that actually are positive, while recall determines the fraction of all positive samples the classification algorithm managed to classify as positive. The main goal when dealing with an unbalanced data set is to increase the recall without hurting the precision (Chawla [9]).

		Predicted Class	
		+	-
Actual	+	TP	FN
Class	-	FP	TN

Table 2.3: Confusion matrix for binary classification problem, adapted from Tan et al. [38]

Accuracy

Accuracy is a metric that summarizes a classification algorithm's overall performance. Given the confusion matrix in Table 2.3 the accuracy is defined as

$$A = \frac{TN + TP}{TN + TP + FN + FP} \tag{2.22}$$

Thus, the accuracy measures the fraction of samples that was correctly classified. When dealing with an unbalanced data set, a high accuracy does not necessarily mean good performance. The minority class is often more important than the majority class for classification with unbalanced data sets. To classify one more positive sample correctly one may have to accept classifying two or more negative samples as positive as well. Thus, even though one are more satisfied with the classification, the accuracy decreases.

Area Under Curve (AUC)

When dealing with unbalanced data sets the limitation of the accuracy as a performance measure was quickly established. ROC curves soon emerged as a popular choice (Hernández-Orallo et al. [16]). The Receiver Operating Characteristics (ROC) curve is a representation of a classifiers performance which takes the TP and FP rates at different discrimination thresholds into consideration. An example of such a curve can be seen in Figure 2.1. The ROC curve is made by representing the $\%FP = \frac{FP}{TN+FP}$ on the X-axis and $\%TP = \frac{TP}{TP+FN}$ on the Y-axis. The ROC curve is swept out by decreasing the discrimination threshold of the classifier - or by undersampling the majority class - moving the ROC point to the upper right in Figure 2.1. An optimal point for an ROC curve would be (0,100) where all positive samples are correctly classified and no negative samples are misclassified.

AUC measures the area under the ROC curve and can be interpreted as the expected true positive rate, averaged over all false positive rates Ferri et al. [14].

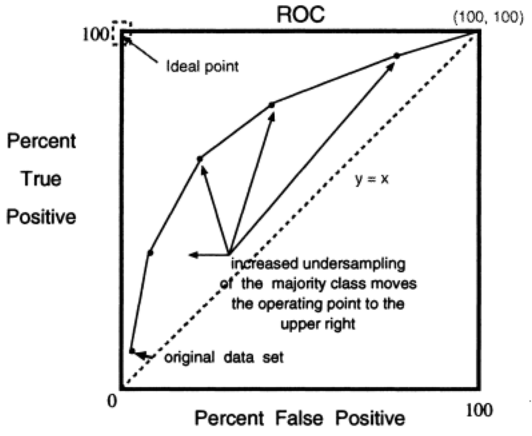


Figure 2.1: An example of an ROC curve, borrowed from Chawla [9]

Brier Score

Brier [7] introduced the brier score, a metric that measures the accuracy of probabilistic predictions. Given a binary classification problem; N samples that you want to classify; and that the classifier outputs the probability of a sample belonging to the positive class; the brier score measures the mean squared difference between:

- The predicted probability of a session belonging to the positive class, n_i
- and the actual outcome, o_i

Thus the lower brier score, the better the probabilistic predictions are. An optimal brier score is 0, where all the probability estimates are correct. In such a situation, among samples that have a predicted probability of x for belonging to the positive class, $100 \cdot x$ percent of the samples should actually belong to the positive class. The brier score is given by the following function

$$BS = \frac{1}{N}\sum_{t=1}^{N}(n_i - o_i)^2 \tag{2.23}$$

2.5 Software

As all the classification algorithms and other methods used in this thesis is implemented in several tools available for free online, none is implemented from scratch. The two machine learning tools used in this thesis is Weka [41], and scikit-learn [33] which is a machine learning tool in Python [29]. All graphs in this thesis are made using *matplotlib*, a library for Python.

2.5.1 Weka

Weka (Waikato Environment for Knowledge Analysis) is a machine learning software developed at the University of Waikato in New Zealand. It provides a graphical user interface with tools for data pre-processing, classification, regression, clustering, association rules and visualization. In order to use the methods provided in Weka, the input data has to be in form of an ARFF (Attribute-Relation File Format) file. The software is written in Java and available for free under the GNU (General Public License).

2.5.2 scikit-learn

Scikit-learn is a machine learning tool for the Python programming language, which provides classification, regression and clustering algorithms and more. It is built on the Python libraries NumPy, SciPy and matplotlib. This software tool is open source and commercially usable under the BSD license.

Chapter 3

Data analysis

In this chapter we will present the data from the RecSys Challenge more detailed. We will give information about the raw data and how we have chosen to pre-process it. Further, we will present an analysis of the data. The analysis aims to reveal important information that we can make use of when classifying session behaviour later on. As we mentioned in Section 1.1.1 we want to figure out what separates a buy-session from a not-buy-session as well as which of the items a buy-session purchases. On this basis we have made two analysis sections: one for the buy-or-not classification task and one for the items-bought classification task. The data analysis was conducted iteratively as we used the information we found to try to improve our score in the RecSys Challenge (see Chapter 4) as we discovered interesting characteristics in the data. The most important findings of our analysis is presented in this chapter. Some additional analysis can be seen in Appendix A.

3.1 Raw data

The raw data was downloaded from the web page of the RecSys Challenge [1]. It was divided in three files, where one of them consisted of click events to be used in the challenge (test data). The two others were for training, and consisted of one file of click events, and one with buy events.

3.1.1 Click events

The files of click events contain comma separated lists of features, where one line of features is one click event. One event consists of the following features: *Session ID*, *timestamp*, *Item ID* and *category.*

- *Session ID* - The ID of the session. A session can contain several clicks. Represented by an integer.
- *timestamp* - The time when the click occurred. Represented by a string formatted in this manner: YYYY-MM-DDThh:mm:ss.SSSZ
- *Item ID* - The ID of the item clicked. Represented by an integer.
- *category* - The category of the item clicked. "S" indicates a special offer; the integers 1-12 represent a real category identifier; "0" represents a missing value; any other integer indicates a brand.

3.1.2 Buy events

The file of buy events contains comma separated lists of features, where one line of features is one buy event. One event consists of the following features: *Session ID*, *price*, *Item ID*, *timestamp* and *quantity*.

- *Session ID* - The id of the session. A session can contain several buy events. Represented by an integer.
- *timestamp* - The time when the buy occurred. Represented by a string formatted in this manner: YYYY-MM-DDThh:mm:ss.SSSZ
- *Item ID* - The ID of the item clicked. Represented by an integer.
- *price* - The price of the item. Represented by an integer.
- *quantity* - How many of this item were bought. Represented by an integer.

The same *Session ID* can contain several buy events with the same *Item ID*.

3.1.3 Training and test data overview

Here we will give an overview of the RecSys data. We will give a short introduction of both the training data and the test data.

Training data

The training data consists of two files. One file containing click events from sessions, and another file containing buy events from these sessions. In the list below you can get a quick overview of the amount of data and some other interesting facts.

- Number of clicks: 33,003,944

- Number of buys: 1,150,753
- Number of sessions: 9,249,729
- Number of sessions that buy something: 509,696
- Number of items: 52,739
- Share of buy sessions: 0,05510388

We observe that the data set is large, containing over 33 million click events. Only 5.5 percent of the sessions end up buying. There are over 50 thousand items and we can see from Table 3.1 that the number of clicks on each item varies a lot.

Measure	Average	Median	STD
Session length (seconds)	382	128	757
Session length (clicks)	3.57	2	3.79
Clicks per item	625.79	22	2810.05
Purchases per session	0.124	0.0	0.687

Table 3.1: Training data overview

Test data

The test data contains a file consisting of click events. The events are gathered from the same time period as the training data. As seen in the list below, the test data contains about one third of the number of sessions in the training data.

- Number of clicks: 8,251,791
- Number of sessions: 2,312,432
- Number of items: 42,155

Measure	Average	Median	STD
Session length (seconds)	383	128	760.4
Session length (clicks)	3.57	2	3.8
Clicks per item	195.7	14	780.9

Table 3.2: Test data overview

As the number of buy-sessions and items bought are unknown in this data set, we have to assume that the characteristics of the training data is representative for the test data. The session length is similar both in number of clicks and in duration. Number of clicks per item is lower, which is natural as this data set contains far less clicks. The number of items in the test set is smaller than in the training set.

3.1.4 Data pre-processing

To get as much information as possible out of the data we have done some pre-processing. The data has some clear deficiencies. The biggest deficiency being that the first 44.4 percent of the events in the click events file have the category feature 0 - meaning no category specified. As we found this unlikely, we constructed a map with *Item ID* as the key and all the categories we could find for this *Item ID* as values. This showed that the same item could have several categories, and that the ones having category 0 in the first part of the data set, usually were represented further down with different categories. We chose to represent an item with a category feature (from 0-12) indicating what category the item is related to most often.

One category in the data set is represented with an 'S', and means that the item is on sale. We modified this by creating a new feature named *sale*, which can hold the values 1, 0 or *None*. As we discovered that the sale category was only represented in a certain time period of the data set (same period as *category*), we figured that it would not make sense to set 0 on *sale* if the current period did not represent sales. Thus, *None* is set for the time period that we implicitly think that sales is not represented.

3.2 Buy-or-not analysis

What makes a buyer a buyer, and what separates a buy-session from a session with no buys? Our goal with the buy-or-not analysis is to reveal different aspects of the data that can tell us whether a session ended up buying or not. This information will be important when we are to classify sessions. Even though the analysis is conducted with data provided by the RecSys Challenge the analysis is rather general and can be applied for similar data sets. We will look closer at aspects like time, the category of the items, the clicks that occurred in a session and which items the session visited. Moreover, we will analyse how these properties correlate - if they compliment each other or if they give more or less the same information.

3.2.1 Time

In order to discover how user behavior affects the probability of a session buying something, we have looked at different aspects of time. This includes the time of when a session is active; when a session buys something; and different measures of duration in each session.

When we have analysed how time affects the probability of a session buying something, we have encountered a possible source of error. The data only tells us when the last click happened and not when the session actually ended. Because of this we can not know how long the session spent watching the last item clicked. As a solution we have chosen to set the end of the session to when the last click happened, this leading to the time spent on the last item being 0 seconds. Most of the graphs displaying time measures are based on a set that consists of 6 percent of the training data. More information about this data set can be seen in Section 4.1.

The first aspect we will look into is when people are active and buy products on the e-commerce site. This is done by analysing the entire time period, using the 6 percent data set. We look at the activity per month, weekday and per hour, in order to see if there are any distinct differences.

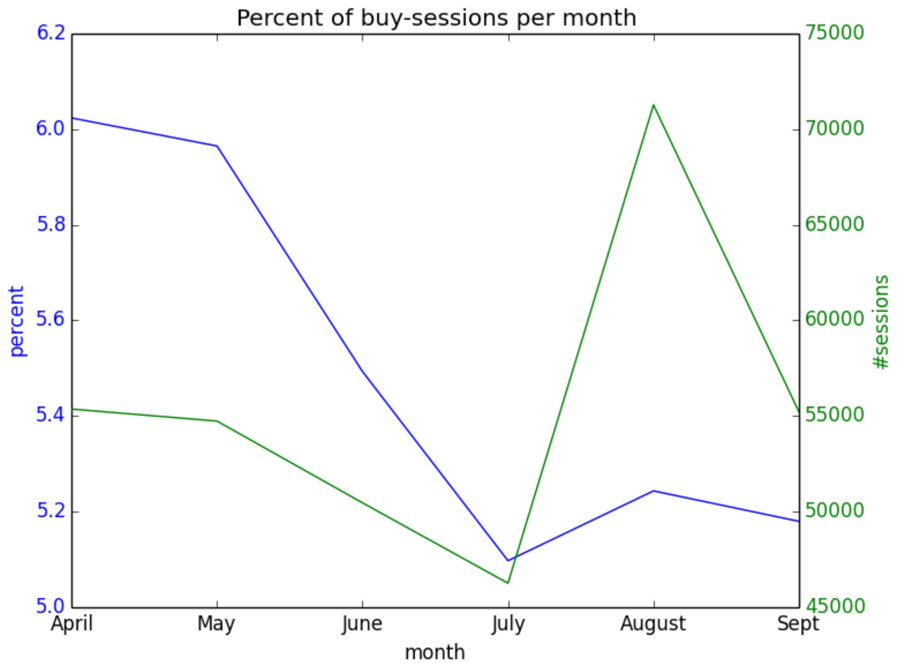


Figure 3.1: Percent of sessions buying, per month

The data consists of user sessions from an e-commerce site in the time period April to September 2014. As seen in Figure 3.1 the number of active sessions have a peak in August, which is the only month that separates noteworthy from the rest. We see that the share of sessions that buys something decreases towards the end, in that it begins with 6 percent in April and ends with 5.2 percent in September.

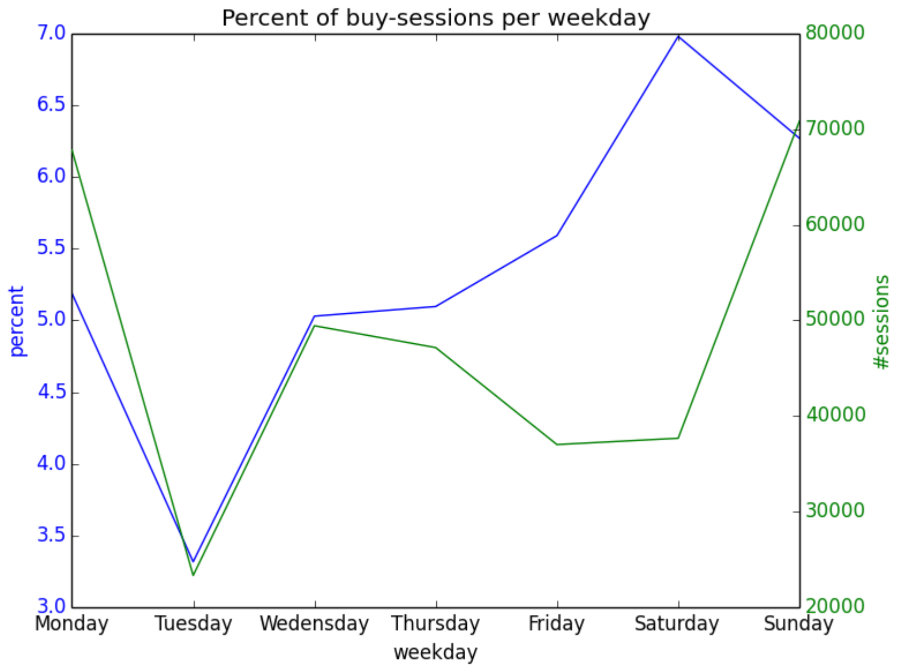
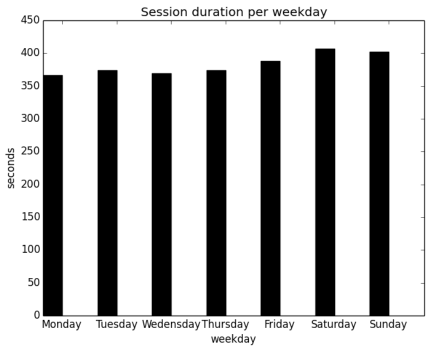
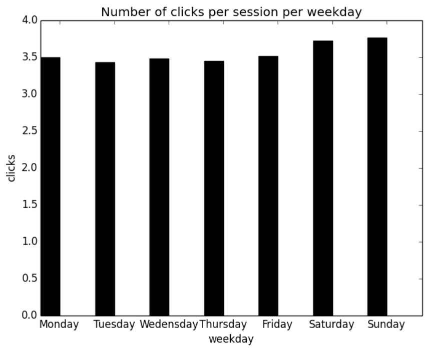


Figure 3.2: Percent of sessions buying, per weekday

Figure 3.2 shows the collected active sessions and share of buyers per weekday. We can see that the share of buyers are higher during the weekend than the rest of the week. A session on a Saturday has over doubled the probability of buying something compared to a session on a Tuesday - 0.070 versus 0.034. The number of active sessions are highest during Sunday through Monday, and at its lowest on Tuesday.



(a) Duration



(b) Clicks

Figure 3.3: Average duration and clicks per session, per weekday

It would be interesting to know why we observe the behaviour we do in Figure 3.2. Are the behaviour in general different on Tuesdays compared to Saturdays, or do people simply shop more on Saturdays regardless of their behaviour? In Figure 3.3 we see plots of the average duration and clicks of a session on the different weekdays. The plots show us that both duration and clicks of the sessions are pretty similar, but a little bit higher on Saturdays and Sundays. As the percent of sessions that buy something on Saturday is twice the percent on Tuesday, and the difference in session duration and clicks are relatively small, this may indicate that there are in general a higher number of buys on Saturdays even though the consumer behavior looks similar. This concept provides valuable information in order to understand consumer behavior, in that people seem more willing to buy on weekdays. A reason might be more free time and a less stressful environment.

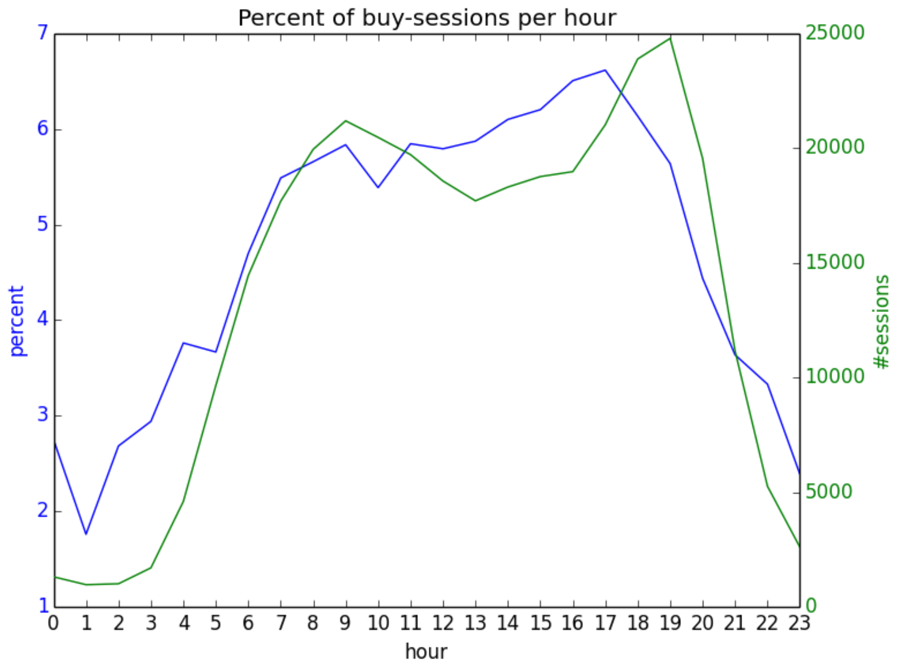


Figure 3.4: Percent of sessions buying, per hour

The last overview of when consumers interact with the e-commerce site is shown in Figure 3.4, where we see the collected behavior during a day. As expected, the number of active sessions are lower during the night. The two curves have a relatively similar trend, both reaching their peak around 5-7 PM. The probability of a session buying is at its lowest 1 AM (approx. 0.018) and its highest 17 PM (approx. 0.066).

Up until now, we have analysed the time of when sessions are active and the percentage of buys at different time periods. The following graphs will illustrate different measures of duration in each session.

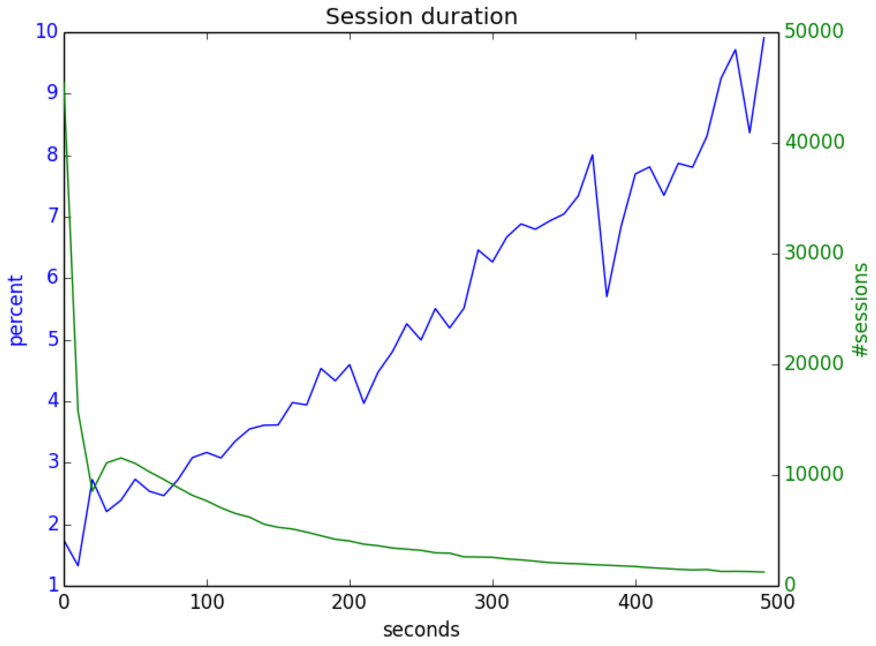


Figure 3.5: Relationship between session duration and percent of buy-sessions

Figure 3.5 shows that as the duration of a session increases, the probability of that session buying something increases. We also observe that the majority of the sessions have a relatively low duration. If a session lasts for 500 seconds we can see from the graph that close to 10 percent of the sessions end up buying, while if a session lasts for 100 seconds the percentage is around 3. The behaviour we observe here is intuitive. One would assume that spending more time means that the session expresses a greater interest in buying something. When the duration is close to zero seconds, the number of sessions rise drastically - and we can see that a lot of the sessions are only active in 0-25 seconds. An explanation for this can be that the easy access to e-commerce has made it popular to briefly browse some products online, with little intention of buying. However, as we can not know what type of events are included in the data, it is possible that the session in fact lasts longer than what the graph displays.

To summarize, different aspects of time affect the probability of a session buying something. The main findings are: as the duration of a session increases, the probability of that session buying something increases; and the probability of a session buying heavily depends on when the session is carried out. It can look like a session carried out on a Saturday has a much higher chance of buying compared to a session on a Tuesday - despite having more or less the same behaviour.

3.2.2 Clicks

As the data we work with is made up by click events, it seemed natural to analyse how the patterns of clicks affected a sessions probability of buying something. This is done by looking at the total number of clicks, and repeated clicks on the same item.

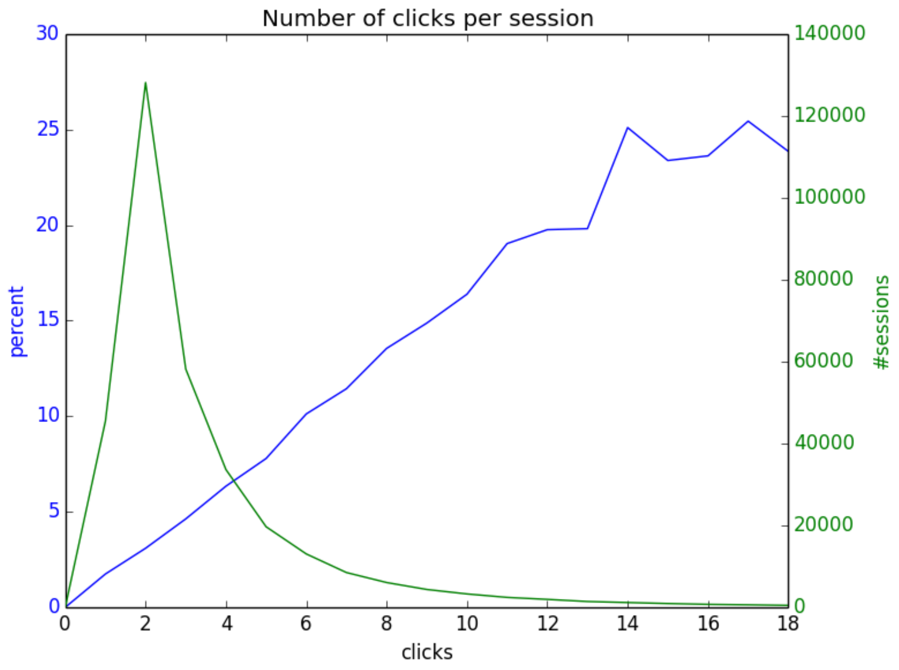


Figure 3.6: Relationship between number of clicks in a session and percent of buy-sessions

In Figure 3.6 we see that the probability of buying increases as the number of clicks increase. It is also shown that the majority of sessions consist of few clicks - in the region 1-5. The highest probability we can infer by number of clicks is approximately 25 percent, for sessions with around 15 clicks or more. Given the fact that only 5.5 percent of the sessions purchases, it is clear that a probability that high is a strong signal for a buy.

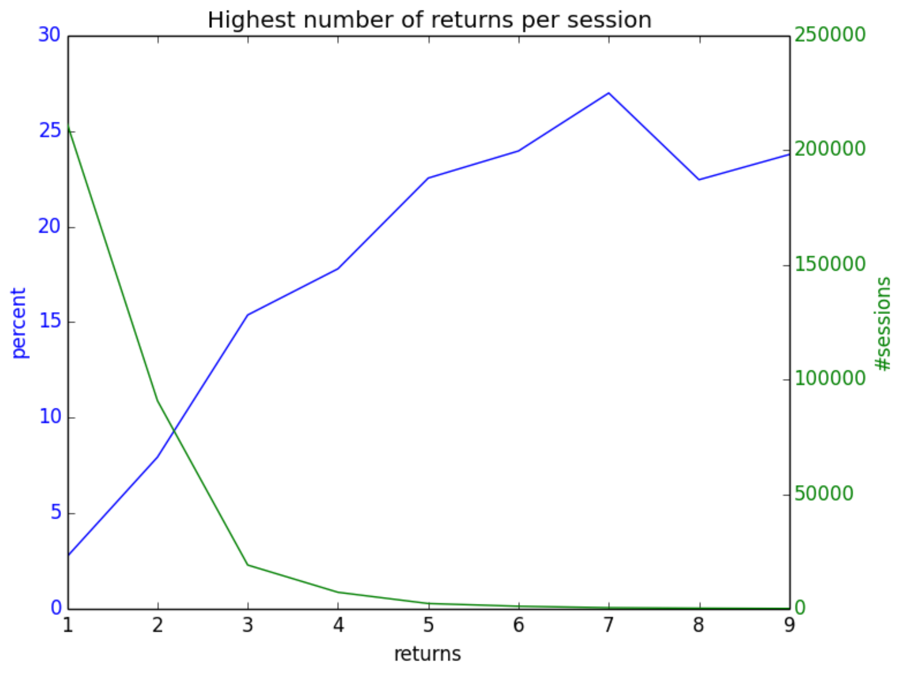


Figure 3.7: Relationship between maximum number of clicks on the same item in a session and percent of buy-sessions

Figure 3.7 displays the maximum number of times a session has clicked on the same item. "1" return means that the session has clicked an item maximum once. Here we see that the majority of sessions only click once on an item, and that the probability of these sessions buying something is low. When a session clicks maximum two times on an item, the probability of buying something is more than doubled, and the probability continues to increase as the maximum number of clicks on an item increases.

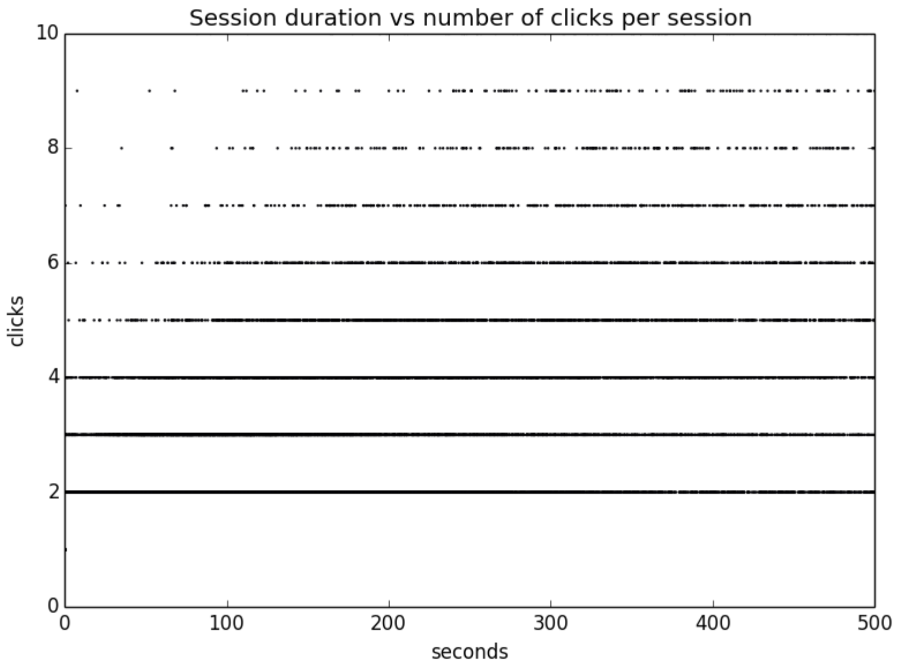


Figure 3.8: Frequency of sessions, given duration and clicks

Figure 3.8 shows a scatter plot of the session duration and the number of clicks. From the plot we see that the number of clicks correlates with the duration of a session. The more clicks a session has, the longer the session lasted. This may mean that the number of clicks does not give as much additional information.

3.2.3 Category

As the raw data contains a feature indicating the category of which an item belong, an analysis of this feature was performed. Possible category values in the raw data is a number from 0-12 indicating category, S indicating that the item is on sale, and any other number indicating the brand of the item. However, after pre-processing this feature now only takes a number between 0-12.

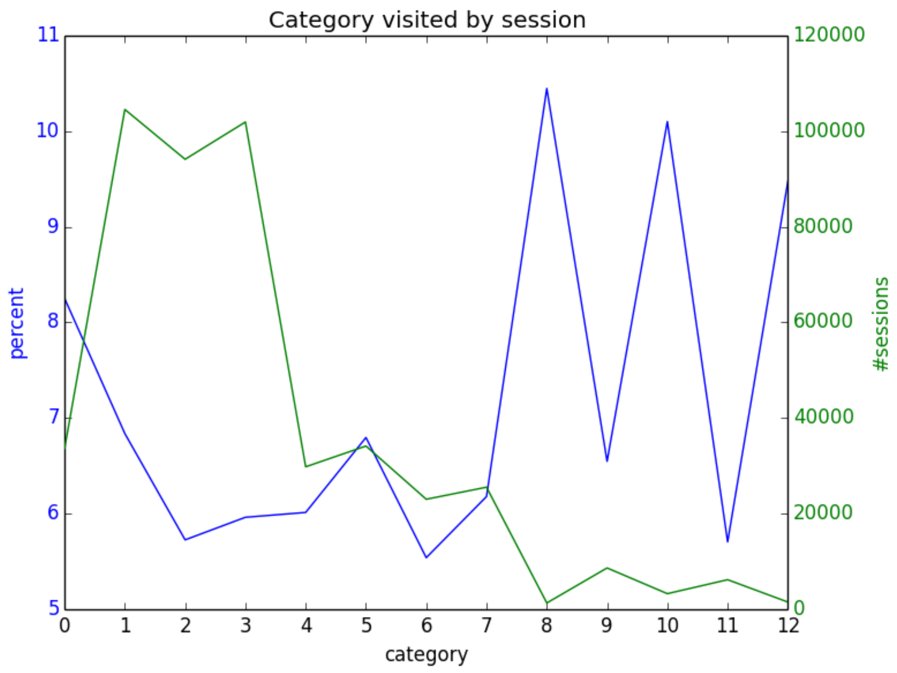
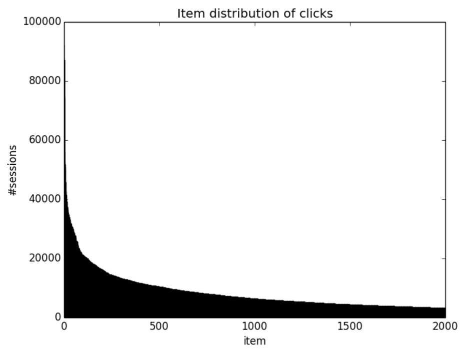


Figure 3.9: Percentage of buy-sessions, given click within category

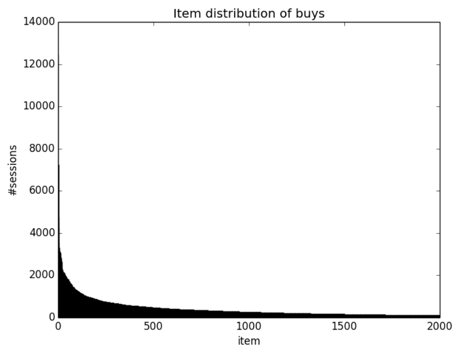
From Figure 3.9 we see that the category with the highest conversion rate is category 8 - meaning that the probability of a session buying is highest when the session has visited this category. It can also be seen that the total number of sessions having visited category 8 is low. This indicate that the category does not typically contain items that consumers enjoy browsing without a purpose of buying. The categories 1, 2 and 3 on the other hand have a large share of sessions clicking items within them, but a lower share of buy-sessions.

3.2.4 Items

In this section we will look at how the items a session visits affect the probability of that session buying. We suspect that there exist some items that have higher probability of being bought when visited than others. Further, there may be some items that when clicked increase the general probability of a session making a purchase.



(a) Distribution of clicks on items



(b) Distribution of buys of items

Figure 3.10: Distribution of clicks and buys over items

In Figure 3.10a we observe that there are a few items that are clicked a lot, and a lot of items that are clicked less. The items included in Figure 3.10a are only the 2000 items that have been clicked the most, this because adding all the items would reduce the readability of the results. Further, we see the same tendency when looking at the frequency of buys per item in Figure 3.10b. Some few items have been bought very often, while the rest of the items have been bought more infrequently. As for clicks, only the 2000 most frequently bought items are included.

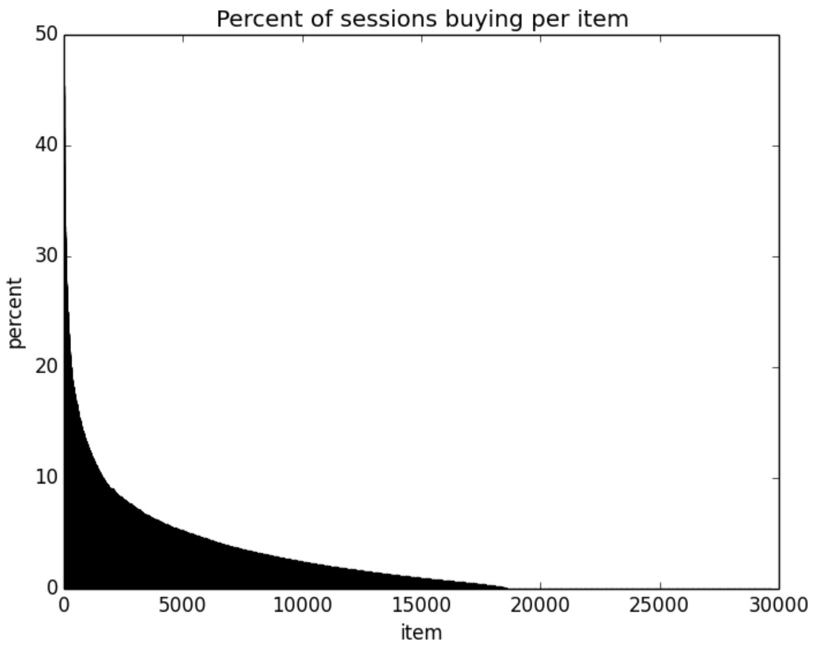
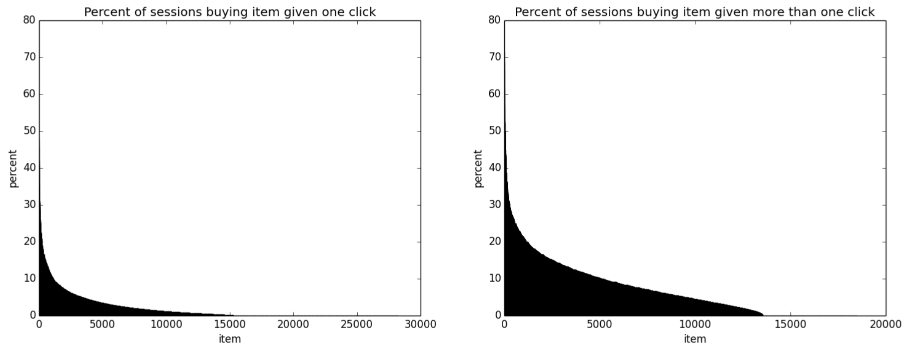


Figure 3.11: Percent of sessions buying item, given item clicked

As Figure 3.10 shows, the number of clicks and buys are not the same for all items. What we can not tell from these graphs is if the same items that are clicked a lot also are bought a lot. If the ratio between number of clicks and number of buys are constant for each item, all items would have the same probability of being bought when clicked. By plotting the probability of buy for each item in Figure 3.11 we see that this is not the case. The probabilities of buying an item varies. Items represented in the graph are only the ones clicked by more than 10 sessions, in order to avoid noisy values.



(a) Percent of sessions buying item, given one click on the item (b) Percent of sessions buying item, given more than one click on the item

Figure 3.12: Percent of sessions buying item, given number of clicks on the item

We ignored the number of clicks a session has on an item when calculating the probability of that session buying that item. We saw in Section 3.2.2 that higher maximum return to an item increased the probability of a session buying something. It would be interesting to see if there exist items which have a low probability of being bought even if the item is clicked twice. In Figure 3.12a we have plotted the probability of a session buying an item given that the session only has clicked that item once. In addition, Figure 3.12b shows the probability of a session buying an item given that the session has clicked that item more than once. We see that an item clicked twice generally has a higher probability of being bought, but that there exist items that are clicked twice that have a relatively low probability as well.

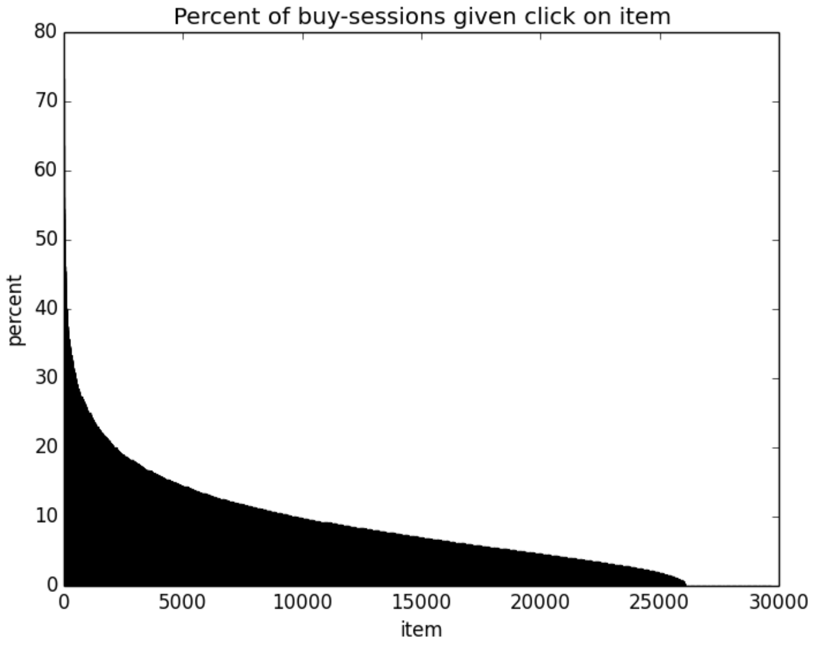


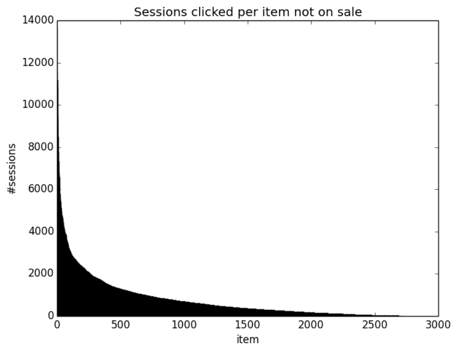
Figure 3.13: Percent of buy-sessions, given click on item

In Figure 3.11 we saw the distribution of probabilities for buying an item, given a click on this item. Figure 3.13 on the other hand shows the probability of a session being a buy-session given a click on a specific item. The underlying idea by analyzing this, is the possibility that if a session clicks a specific item, this increases the general probability of buying something. We observe that some items when clicked give a higher probability of a session ending up buying. This is most likely related to the probabilities of buying a specific item, shown in Figure 3.11.

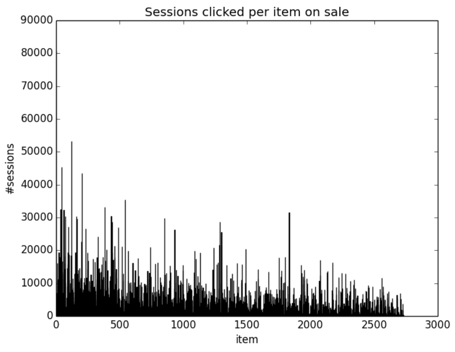
As mentioned in Section 3.1.4, the data contains a category feature, where one possible category is S which means the item is on sale. When doing preprocessing we changed this into a feature named *sale*, which is either 1 for sale or 0 for not sale. We discovered that there seems to be one period where there are sales, and one period where there are no sales at all. We will only include the sale period when analysing sales.

In order to research how sales affect the consumer behavior and purchases, we made two sets of graphs (see Figure 3.14 and 3.15). Figure 3.14a displays the number of sessions that has clicked the items when they were not sale, and

Figure 3.14b shows the number of sessions that has clicked on the same items when they were on sale. Figure 3.15 contains two graphs, where one displays the probability of buying an item that were clicked when not on sale, and the other displays the probability of buying an item clicked when on sale. Both graphs displaying items on sale has the items in the same order as the ones displaying items not on sale.

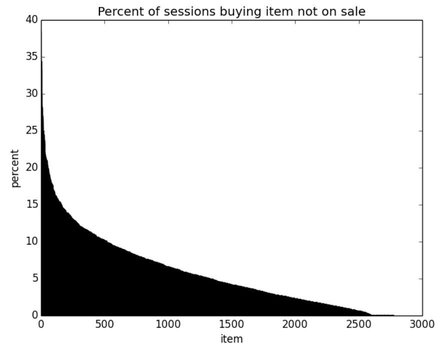


(a) Distribution of sessions clicked, over items not on sale

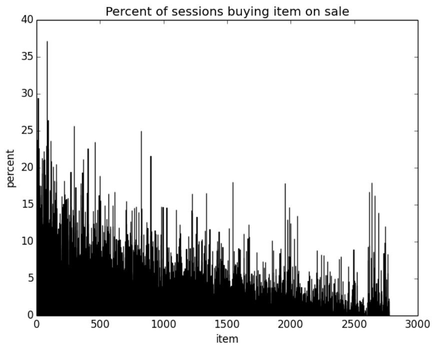


(b) Distribution of sessions clicked, over items on sale

Figure 3.14: Distribution of sessions clicked, over items on sale and not on sale



(a) Percent of sessions buying item, given click on item not on sale



(b) Percent of sessions buying item, given click on item on sale

Figure 3.15: Percent of sessions buying item, given click on item on sale and not on sale

As one can see in Figure 3.14 there are a higher amount of clicks per item in general when the items are on sale. We also observe that some items that are clicked rarely when not on sale, suddenly are clicked a lot when on sale. If nothing else, it shows that putting an item on sale could generate a lot of clicks on it.

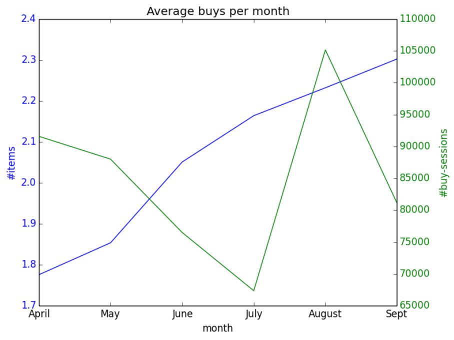
In Figure 3.15, the probabilities of buying each item, clicked by over 10 sessions, for both items not on sale and items on sale, are displayed. We see the same trend as in the click distributions, that items which are not popular when not on sale, suddenly can have a probability of being bought similar to the more popular items when on sale. These graphs show that items on sale do not only generate more clicks, but also give a higher probability of being bought.

3.3 Items-bought analysis

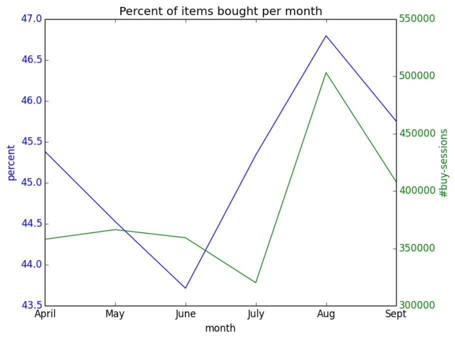
In this section we will analyse the data to find out what makes a buy-session buy a given product. We will look into features like time, number of clicks and the probability of buying an item when the item is clicked. When analysing we are only using the clicks from sessions that ended with a purchase.

3.3.1 Time

We saw in Section 3.2.1 that the time of a session heavily influenced whether the session ended up buying or not. We observed that a session carried out on a Tuesday only has half the probability of buying compared to a session carried out on a Sunday. Further, the probability of buying is slightly higher at the start of the period the data is collected from. Here we want to investigate how time influences the number of items a session buys.



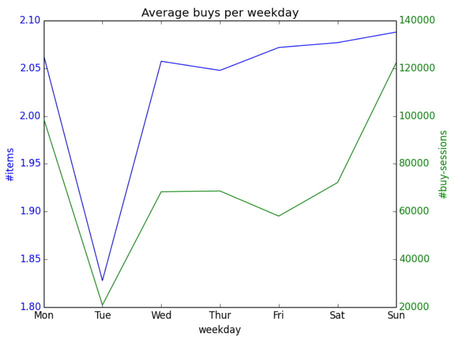
(a) Average number of buys, per month



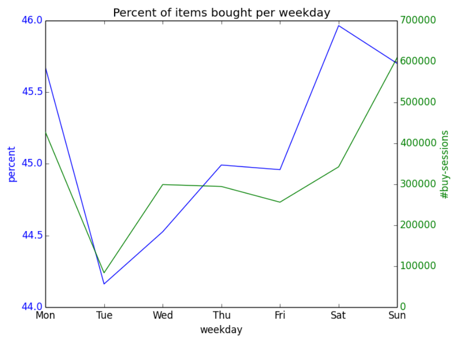
(b) Percent of items clicked that are bought, per months

Figure 3.16: Behaviour over months for buy-sessions

In Figure 3.16a we see that the average number of items a session buys grows over time. In April the average is slightly below 1.8 while at the end of the period, in September, it has grown to above 2.3. It is difficult to say why we observe this kind of behaviour. One possible explanation can be that the online store has improved their design and/or the recommendations the system gives to the consumer. Another explanation can be the time of the year. It could be that people tend to shop more items during the summer season. In Figure 3.16b we have plotted the share of items bought compared to items clicked by a session. The share of items bought is more stable than average number of items bought, and only differ approximately 7 percent at the most. Thus, sessions tend to click on more items later in the period - substantiating the theories about improvements of the site and season variation.



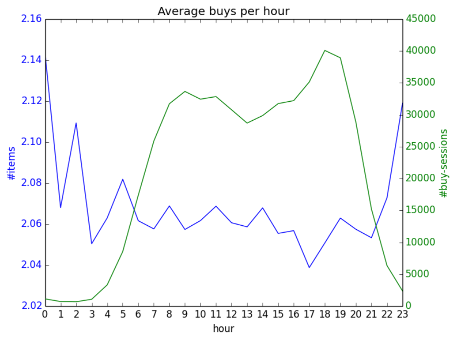
(a) Average number of buys, per weekday



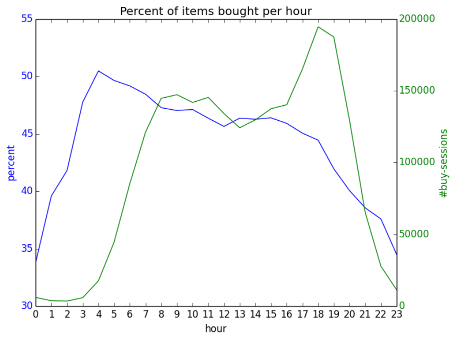
(b) Percent of items clicked that are bought, per weekdays

Figure 3.17: Behaviour over weeks for buy-sessions

Figure 3.17a shows how the average number of buys varies at the different weekdays. We observe that the average is at its lowest on Tuesdays and at its highest on Sundays, with 1.84 and 2.08 respectively. Figure 3.17b shows that consumers purchase more of the items they click during the weekend and Monday compared to rest of the week. However, the differences are small - comparable with the observations made for months.



(a) Average number of buys, per hour



(b) Percent of items clicked that are bought, per hour

Figure 3.18: Behaviour over hours for buy-sessions

Lastly we want to look at how the number of buys for a buy-session varies with the time of the day. In Figure 3.18a we see that the average number of items a session buys is pretty similar for all times of the day. The highest average we observe is from 12 AM to 1 AM where the average is close to 2.14 items. The lowest we observe is 2.04 items at 5 PM to 6 PM. In Figure 3.18b we observe that the percent of items bought has a peak around 4 AM with 50 percent, and decreases throughout the day ending at approximately 35 percent at around 12 AM. Thus, sessions click on more items later on the day.

To summarize, we have seen that the share of items bought varies little over time, as well as for the weekdays. The number of items a session clicks increases towards the end of the period, and also throughout a day.

3.3.2 Clicks

The number of clicks proved to be an important factor when deciding whether a session bought or not. Does the number of clicks on an item also influence the probability of a session buying that item? It is likely to believe. In this section we introduce the term *session-item*. A session-item is a sample consisting of session ID, item ID and features describing all the interactions the session has with this item - e.g when a session has clicked an item two times this is represented as a session-item. We will explore how clicks affect a session-item to end up being a purchase of the item represented.

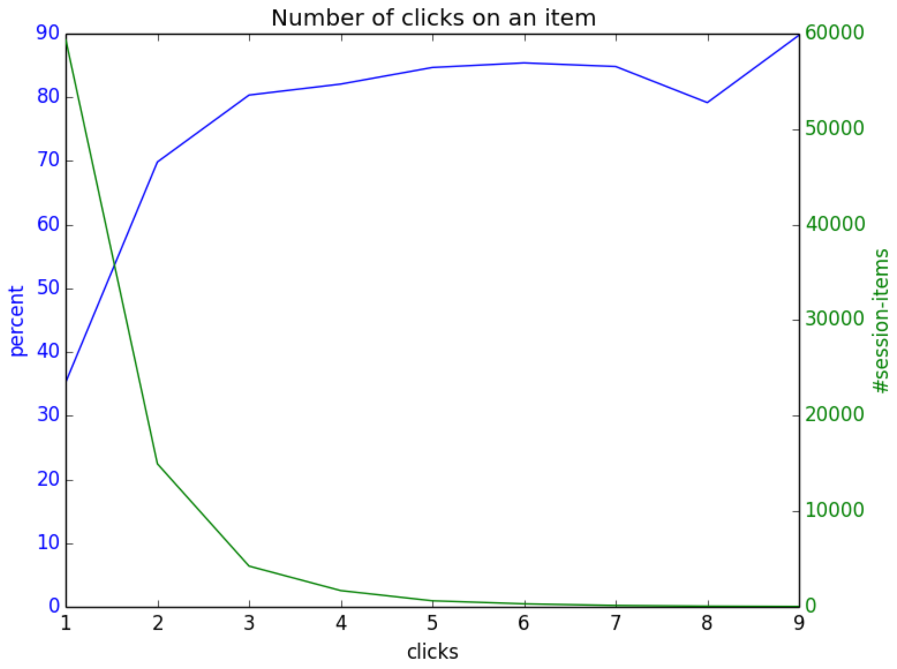


Figure 3.19: Percent of items bought, given number of clicks on the item

In Figure 3.19 we see that there is a clear connection between the number of times an item is clicked and the probability of a session buying that item. The probability is at its lowest if the item is only clicked once and increases as the number of clicks increases. If an item is clicked once, 35 percent ends up being bought. When an item is clicked four times or more the percentage increases above 80. We also notice that clicking an item more than once doubles the probability of that item being bought. The results that Figure 3.19 is showing are intuitive. If a session returns to an item more than once one would assume that the user is expressing interest in the product.

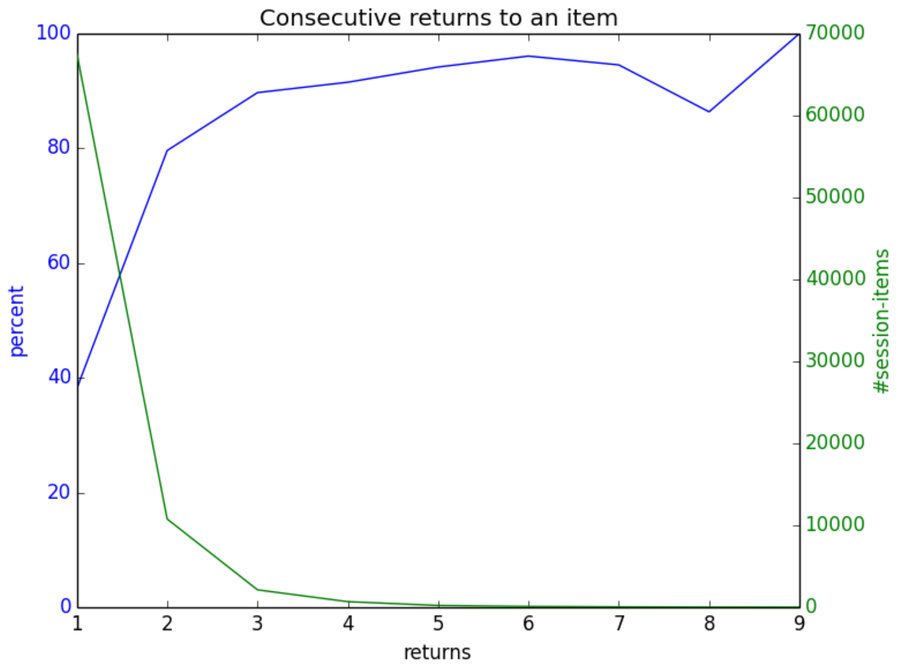


Figure 3.20: Percent of items bought, given consecutive returns to the same item

In Figure 3.20 we observe that when a session repeatedly clicks the same item, without clicking an other item in between, the probability of that item being bought increases. ”1” return means that the item has only been clicked once by the session. If the item is not clicked repeatedly the percentage of items bought is around 40, while when this occurs the percentage increases to over 80. Repeated clicks strongly indicates a buy, but the behaviour is not that common. Only 17 percent of the session-items have repeated clicks. The behaviour we observe in Figure 3.20 is also intuitive by the same argument as for number of clicks. However, clicking the same item twice in a row, without anything else happening in between, seems strange. Since we do not know anything about the online store providing the data it is difficult to say why we observe this behaviour. We can only accept that repeated clicks on an item increases the probability for a buy.

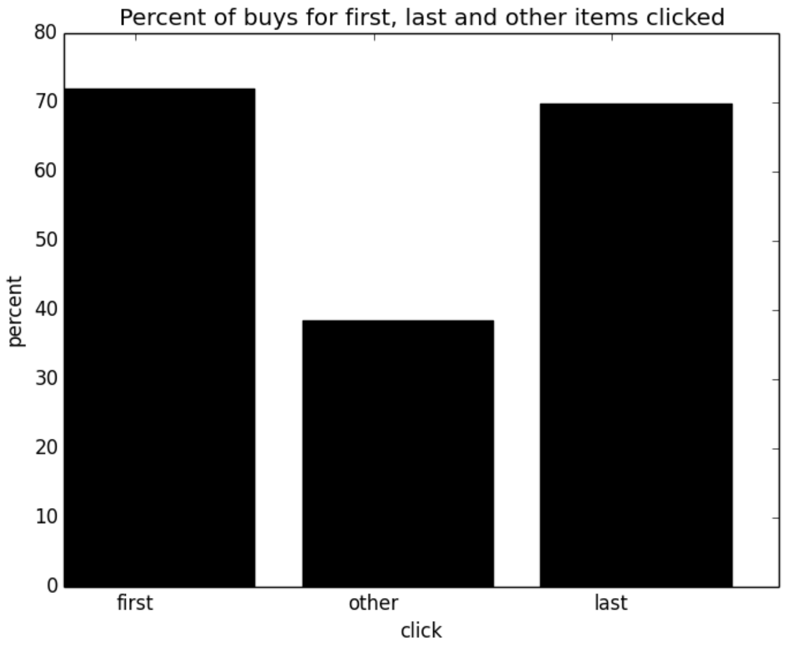


Figure 3.21: Percent of items bought, given item clicked first, last or neither - in a buy-session

Figure 3.21 displays the percent of buys for items clicked either first, last or somewhere in the middle in a buy-session. As seen, the items clicked first or last have a higher probability (approx. 0.7) of being bought compared to the items clicked in between (approx. 0.4).

3.3.3 Category

While the share of buy-sessions per category was analysed in Section 3.2.3, the share of session-items being a purchase per category will be analysed here.

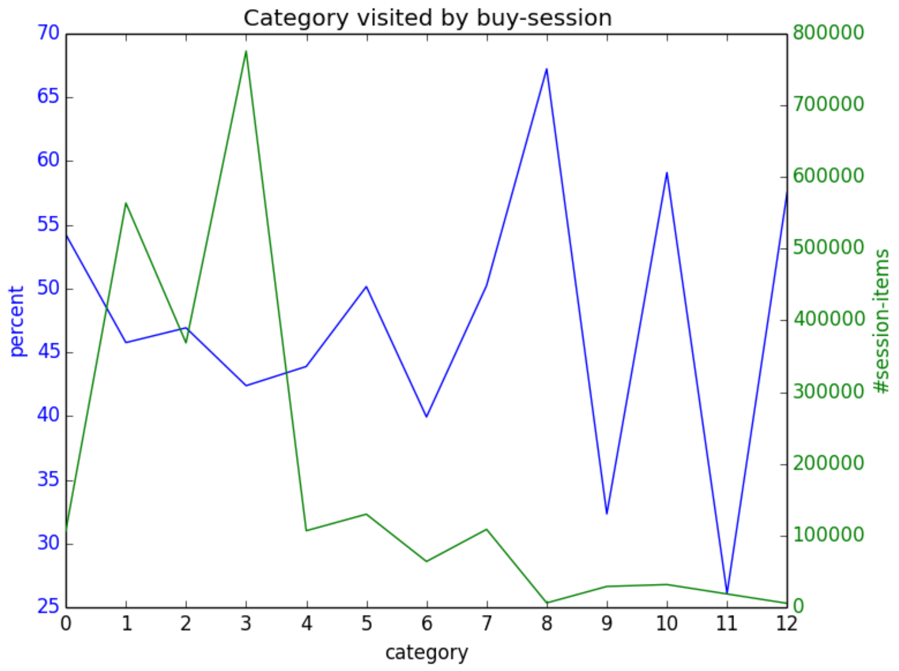


Figure 3.22: Percent of items bought, given category of the item clicked

Figure 3.22 shows the share of session-items within each category being a purchase. In addition the total number of session-items within each category is displayed. The percent of buys is highest for category 8, but one can see that the number of sessions that clicked within this category is low compared to the number of sessions that has clicked on categories 1, 2 and 3. For these three categories we see a rather similar percentage of buys, all around 45.

3.3.4 Items

In Section 3.2.4 we saw that there are items that are bought more often than others when clicked, when considering the clicks from all sessions. One would assume that the same also holds when only looking at buy-sessions.

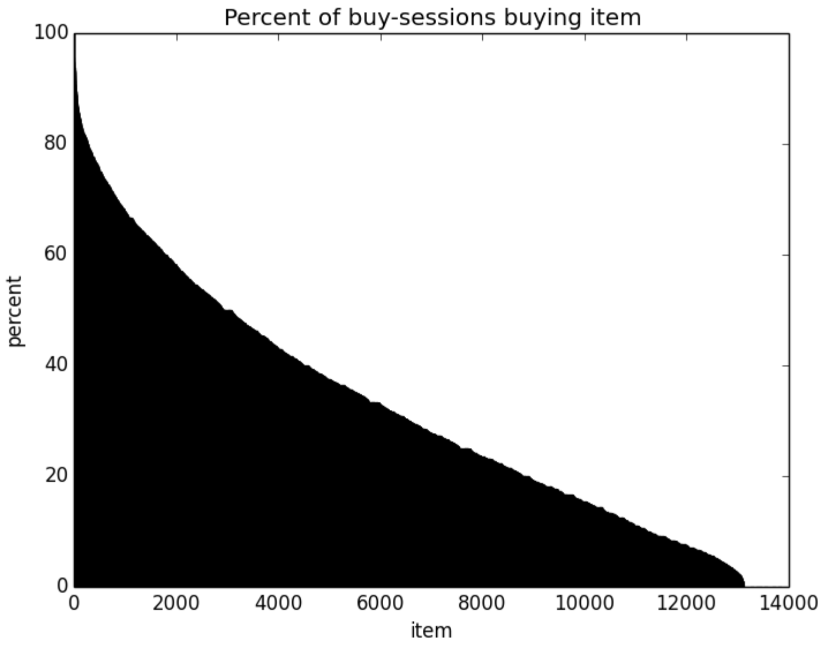
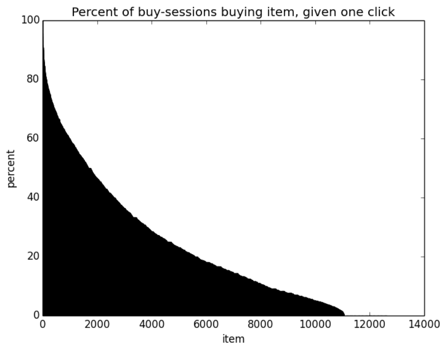
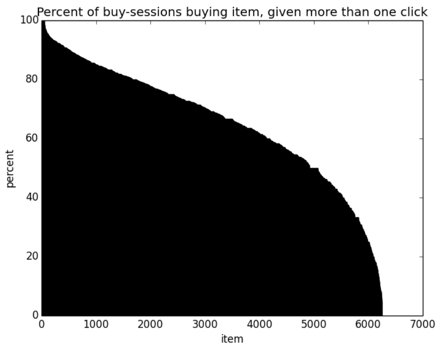


Figure 3.23: Percent of buy-sessions buying item, given click on the item

In Figure 3.23 we see the percent of buy-sessions buying a specific item given that the session has visited the item. The probability of buying an item is higher for buy-sessions compared to when looking at all sessions (see Figure 3.11), which is intuitive. We observe that there are some items that are bought more often when visited than others.



(a) Percent of buy-sessions buying item, given one click on the item

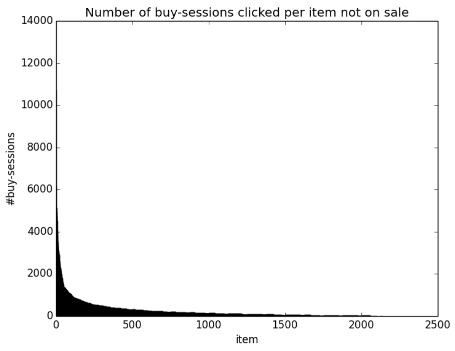


(b) Percent of buy-sessions buying item, given more than one click on the item

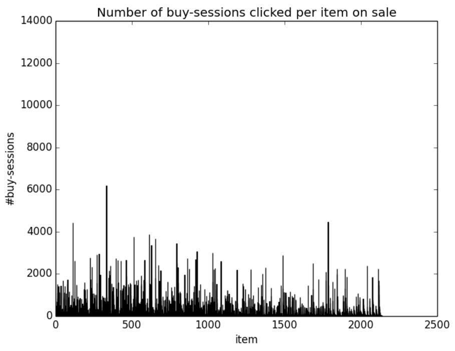
Figure 3.24: Percent of buy-sessions buying item, given number of clicks on the item

Further, we see in Figure 3.24a that there exists some items that are bought often even when clicked only once. Figure 3.24b shows that most of the items are bought over 50 percent of the time when clicked twice or more. Moreover, if we compare 3.24a and 3.24b we see, as expected, that clicking an item more than once increases the probability of that item being bought - which we also observed in Section 3.3.2.

As in Section 3.2.4 we want to analyze the affects of sale, but now for sessions that buy. This is done in the same manner, by comparing a graph with click distribution over items not on sale versus the distribution over items on sale, and by comparing the probability of buy over items on sale and items not on sale.



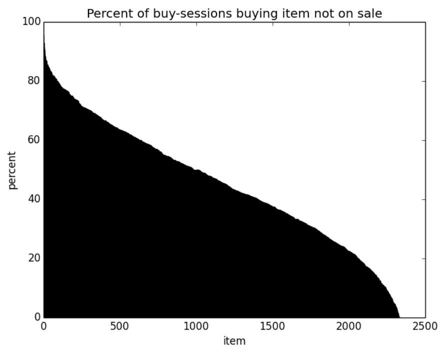
(a) Distribution of buy-sessions clicked, over items not on sale



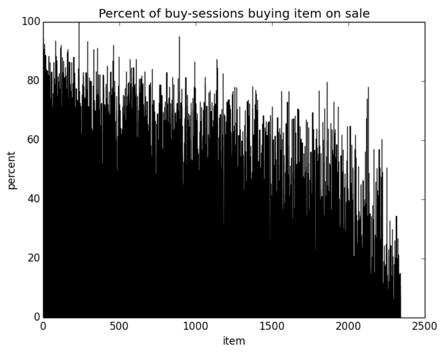
(b) Distribution of buy-sessions clicked, over items on sale

Figure 3.25: Distributions of buy-sessions clicked, over items on sale and not on sale

Figure 3.25b is sorted in the same order of items as 3.25a in order to see the difference in how frequent items on sale are clicked compared to items not on sale. As described in Section 3.2.4, only the sale period is analysed. What we see from Figure 3.25 is that the number of clicks on sale items are more spread than on the items not on sale. For the items not on sale we can see a typical long tail, in that a low share of the items are frequently clicked, and the majority of items have almost no clicks. When items are on sale it clearly reduce the long tail pattern, in that less popular items are clicked more frequently. It is shown that the popular items from 3.25a still have a high number of clicks, also when they are on sale.



(a) Percent of buy-sessions buying item, given click on item not on sale



(b) Percent of buy-sessions buying item, given click on item on sale

Figure 3.26: Percent of buy-sessions buying item, given click on item on sale and not on sale

Figure 3.26b shows that the probability for each item in general grows when the items are on sale. If we look at the items around 6000, the probability for items not on sale is around 40 percent, while when these are on sale the probability for several of them is between 60 to 80 percent. The highest benefit of sale seems to be that rather unpopular items suddenly have a potentially higher probability of being both clicked and bought.

Chapter 4

Process and results

In this chapter we will present the process we went through and the results we got up until our final solution in the RecSys Challenge. As mentioned earlier we are using the challenge as an indicator for how well we can predict a consumer's behaviour, and in each step of the process we will provide a score telling how much improvement we have achieved, if any. We have divided the process into eight iterations. Each iteration is represented as a section where our approaches and results are presented. The first seven iterations are about finding the best approach, including: which classification algorithm to use; how to use the classification algorithm; and what features to use. The last iteration is geared towards optimizing our solution towards the scoring-function in the RecSys Challenge.

4.1 Local computation of scores

There is a limited number of times per day one can upload a solution to the challenge. This combined with the time consumption of producing a solution have yielded the need of a local testing environment. We have randomly selected 6 percent of the sessions from the training data handed out and split it into our own training and test data. From now on, when referring to training data and test data, we are referring to these constructed data sets. The training data constitutes 60 percent of the randomly selected data (3,6 percent of the entire data set), while the test data constitutes 40 percent (2,4 percent of the entire data set). We are using the scoring-function, as explained in Section 1.1.1, when computing a score in our local test environment. The scores presented in this chapter will both be results from our local testing environment as well as from the RecSys Challenge. A score from our local testing environment will be referred to as *local score* (LS) while a score from the RecSys challenge will be referred to as *challenge score* (CS).

In Table 4.1 we give an overview of the training data we have used to build classification models in the first six iterations.

Measure	Value
Number of clicks	1188128
Number of sessions	333310
Number of sessions that buy	18327
Buy sessions/Sessions ratio	0,0549
Buys per session	0,122

Table 4.1: Local training data overview

We can see from Table 4.1 that the training data is relatively representative for the total data set (see Section 3.1.3), in that the ratio between number of buy-sessions and all sessions only deviate 0.3 percent, and that the measure of buys per session deviate 1.6 percent. Further, we ran local tests with Naive Bayes to see if the amount of training data was sufficient. The results can be seen in Table 4.2. We observe that the results do not change drastically until the training set is reduced to 20 percent of its original size (0.72 percent of the total data set). However, some classifiers require more data to learn a good model than Naive Bayes and the need of data will increase as the number of features increases. In Iteration 7 and 8, where we build classification models for different time periods, we will upload solutions using all the original training data. The amount of data used for each classification model depends on the length of the time periods.

	Buy or not			Items bought			
Fraction of training data	P	R	ROC	P	R	ROC	LS
1	0.167	0.235	0.733	0.108	0.243	0.547	**536**
0.8	0.167	0.236	0.732	0.108	0.242	0.546	535
0.5	0.167	0.236	0.733	0.108	0.241	0.546	530
0.2	0.167	0.232	0.733	0.108	0.241	0.546	509

Table 4.2: Local testing environment with different fractions of (local) training data using Naive Bayes

An overview of the test data, used for computing local score, can be seen in Table 4.3. As for the training data, the test data looks to be representative. We will use the local score computed on this test set only as a guideline, as the amount of data is rather small compared to the challenge data. When making important and tight decisions we will use the challenge score. Tight meaning that the difference in local score is small.

Measure	Value
Number of clicks	793105
Number of sessions	222279
Number of sessions that buy	12292
Buy sessions/Sessions ratio	0,0552
Buys per session	0,124

Table 4.3: Local test data overview

The maximum score given a test set of clicks in the RecSys Challenge can be computed as:

$$Score_{\max} = |S_{b_{\text{test}}}| \cdot \left(1 + \frac{|S_b|}{|S|}\right) \tag{4.1}$$

where $S_{b_{\text{test}}}$ is the number of buy sessions in the test set; S_b is the number of buy-sessions in the total training set; and S is the number of sessions in the total training set. In a similar manner the minimum score can be computed as:

$$Score_{\min} = -\left(|S_{\text{test}}| - |S_{b_{\text{test}}}|\right) \cdot \frac{|S_b|}{|S|} \tag{4.2}$$

Thus the maximum local score for our test set is $12292 \cdot (1 + 0.05510388) \approx 12969$ and the minimum is $-(222279 - 12292) \cdot 0.05510388 \approx -11571$. For the challenge data set the maximum score is estimated to $(2312432 \cdot 0.05510388) \cdot (1 + 0.05510388) \approx 134446$, and the minimum to $2312432 - (2312432 \cdot 0.05510388) \cdot 0.05510388 \approx -120402$

4.2 Evaluation method

During the process all approaches have been evaluated. This includes choice of algorithms, algorithmic parameters and features. This section will cover the evaluation method for items-bought classification and buy-or-not classification, in that order.

4.2.1 Items-bought classification

To provide information about the features extracted for items-bought classification we have included two measures: information gain and information gain ratio (see Section 2.4). In situations where we have constructed a feature in multiple ways, and need to decide which version to continue using in the next iterations, we will use local score as an evaluation measure. An example of such a feature is the probability of a session buying a certain item.

When evaluating an algorithm we looked at the local score of the algorithm when using all features extracted up to this point. We computed the local score by testing items-bought classification on all buy-sessions in the test data. The drawback with this measure is that we do not know exactly how the scoring-function weights precision and recall. One algorithm may give a high recall and a low precision, while another may give a high precision, but a rather low recall. The algorithm with the high recall may score better than the one with the high precision. However, it could be that lowering the threshold for accepting a session as a buy-session for the algorithm with the high precision, could have increased the local score, maybe also above the other algorithm. Therefore we have decided to also include precision and recall. Area under ROC curve (referred to as ROC), which measures a classification algorithm's performance at different discrimination levels, will be included by the same argument. The score will be the major evaluation measure, but precision, recall and ROC will be included to enlighten the potential of a classifier.

4.2.2 Buy-or-not classification

Same as for items-bought classification, information gain and information gain ratio are presented for the features used in the classification process. We will use local score where we have constructed a feature in multiple ways and need to decide which version to continue using in the next iterations.

When testing an algorithm we have evaluated it by: recall, precision, ROC and local/challenge score. The local/challenge score is computed by using the best items-bought classification we found in this iteration on the session-items (see Table 4.8) produced by the buy-or-not classification. Precision, recall and ROC are included for the same reasons as explained in the previous section.

4.2.3 Optimization process summary

The process for finding the best combination for items-bought prediction and buy-or-not is thus as follows:

1. Use only buy-sessions

2. Learn items-bought by evaluating the local score (precision, recall and ROC).

3. Learn buy-or-not classification from the training set and evaluate on local/challenge score (precision, recall and ROC) - using the best items-bought model from step 2.

This process could be extended by looping step 2 and 3 after the first round. By doing this, one could maybe optimize the interaction between the two sub tasks even better.

Further, one could have implemented an algorithm that optimized its behaviour towards the RecSys scoring-function. The drawback with such an approach is that it is time consuming, thus restricting the number of approaches one has time to try. The advantage is that the algorithm would have been optimized towards the scoring-function instead of some other metric - making it optimized for its purpose. Further, other evaluation measures would have been superfluous.

4.3 Iteration 1: Buy or not focus

The data is highly unbalanced and therefore not suitable for certain classifiers, such as decision trees, without doing modifications to the training data. Decision trees classify a sample by looking at the majority class in the leaf node. When one class dominates the other class, decision trees tend to classify more or less all samples as the majority class. In the first iteration we tested different methods on our unbalanced data set, including decision trees. As explained earlier, we have split the problem into two classification tasks. The first task is to decide if a session made a purchase, and the second is to classify the items a session has visited as bought or not. Our focus in this iteration was to find a starting point for classifying sessions as buy or not-buy sessions. This to have a self produced set of sessions classified as buy-sessions which we could perform items-bought classification on, and produce results that we could upload to the RecSys Challenge.

4.3.1 Items-bought

As the focus in this early phase was to predict whether a session ended up buying something or not, we only applied a simple rule when classifying the items a session has visited as bought or not. The rule was to classify an item as bought if it was the first item clicked by a session, and all other items as not bought. We saw in Figure 3.21 that this should give a rather high precision.

Feature	Type	Values	IG	IGR	P	R	ROC	Score
first	Nominal	0 or 1	0.0594	0.0773	0.722	0.351	0.619	6711

Table 4.4: Iteration 1: Items-bought classification

4.3.2 Buy or not

As explained in Section 2.2.1 a way of representing the data when doing classification is needed. In this iteration we represented each session as a sample and made features related to the session. The data format can be seen in Table 4.5. This format will be used in all iterations for buy-or-not classification, except Iteration 3.

sessionID	duration	clicks	avg_clicks	max_return	max_time	avg_time
11	783,761	12	1.33	3	385,217	65,31

Table 4.5: Iteration 1: Sample format, buy-or-not classification

The analysis showed that features related to time and number of clicks correlated well with the class of the samples. Our analysis of these aspects can be seen in Section 3.2.1 and Section 3.2.2 respectively. These form the basis of the features we extracted in the first iteration. An overview is given in Table 4.6.

feature	Type	Value	IG	IGR
clicks	Numeric	1 - ∞	0.0225	0.0083
avg_clicks	Numeric	1 - ∞	0.0232	0.0110
duration	Numeric	0 - ∞	0.0235	0.0077
max_time_between_clicks	Numeric	0 - ∞	0.0172	0.0056
avg_time_per_click	Numeric	0 - ∞	0.0147	0.0050
max_return	Numeric	0 - ∞	0.0221	0.0162

Table 4.6: Iteration 1: Extracted features, buy-or-not

For each feature, the information gain (IG) and information gain ratio (IGR) are calculated, which can be read about in Section 2.4.1. *Clicks* is simply the number of clicks a session does; *avg_clicks* is the average number of clicks per item; *duration* is the total duration of a session; *max_time_between_clicks* is the maximum duration between two clicks in a session; *avg_time_per_click* is the average duration spent looking at an item, thus the average duration between clicks; and finally the *max_return* is the maximum amount of times a session has clicked on the same item. One would expect the features related to clicks to be correlated, more clicks mean a higher *average_clicks* and a higher *max_return.* The same also holds for the features related to time. Moreover, we saw in Figure 3.8 that the duration of the session and the click feature correlated.

4.3.3 Classification

We tested four classification algorithms in the first iteration: Bayesian Network (BN), Logistic Regression(LR), Decision Tree (DT) and Naive Bayes (NB). The Bayesian Network is built using Weka, while the other models are built with the Python package scikit-learn (see Section 2.5). The default parameters used can be seen in Appendix B. The parameters listed there are used when not specified otherwise.

One of the main challenges for classifying this data set is that it is highly unbalanced, in that the vast majority of the sessions are not buy-sessions. The features we have extracted so far are also most likely dependent of each other. These characteristics are important when choosing a classifier. The Naive Bayes classifier assumes conditional independence between the feature, given the class, which may not be the case for the features we have extracted. Thus, we do not expect the Naive Bayes classifier to produce the best results. As mentioned before the Decision Tree will probably not handle the highly imbalanced data set well and classify most of the samples as the majority class.

Table 4.7 displays a comparison of the classifiers we tested with the features extracted previously in this section. In addition to local score for the end-to-end test; precision, ROC and recall is presented for both items-bought classification and buy-or-not classification.

	Buy-or-not			Items-bought			
Classifier	Precision	Recall	ROC	Precision	Recall	ROC	LS
BN	0.174	0.406	0.750	0.113	0.254	0.546	**842**
NB	0.166	0.234	0.733	0.108	0.242	0.548	537
DT	0.126	0.130	0.498	0.084	0.261	0.525	116
LR	0.298	0.016	0.739	0.160	0.147	0.624	40

Table 4.7: Iteration 1: End-to-end classification

We see from the results that the Decision Tree most likely did not handle the imbalanced data set. Further, we observe that Naive Bayes scored well in practice despite our assumption about dependence. Logistic Regression has a high precision, but a rather low recall. Bayesian Network and Naive Bayes outperforms the two other algorithms when it comes to local score. As mentioned earlier the scoring-function may appreciate a high recall above a high precision. Most likely lowering the threshold for accepting a session as buy-session for Logistic Regression would have increased its local score. The Bayesian Network classifier has a high recall compared to Naive Bayes and Logistic Regression, but at the same time also a good precision.

Later in this chapter we will work on solving the problem with unbalanced data (see Section 4.6). In Section 4.8 we will also perform feature selection to see if Naive Bayes can perform even better when given less dependent features. In the two next iterations we will continue using Bayesian Network to improve and find new features, as this classifier worked best on the data set in terms of the local score metric for buy-or-not classification. When we ran the classification on the challenge data and uploaded the result we got a score of 12018.4.

4.4 Iteration 2: Items

In the first iteration we got a starting point for how to predict if a session ended up buying by looking at the duration and the clicks of a session. In this iteration we look at the specific items a session has visited and how we can make use of this when predicting if a session buys, and also when predicting what the session purchased. We will first present our approach for predicting items bought and then present new aspects for buy-or-not afterwards. Lastly, we will run classification for both items-bought and buy-or-not to see if we have done any improvements with our new features.

4.4.1 Items-bought

In the first iteration we prioritized finding an approach for predicting if a session bought. In this iteration we used a Bayesian Network to predict what sessions buy as well. Other classification algorithms will be tested in later iterations. We chose to represent each item a session has visited as a sample and make features related to that *session-item*. Examples of the data format can be seen in Table 4.8.

sessionID	itemID	last	first	Item_duration	n_clicks	prob
11	48677876	1	0	4.73	2	0.73
11	48764443	0	1	8.93	1	0.20

Table 4.8: Iteration 2: Sample format, items-bought

The features we have extracted in this iteration is presented in Table 4.9. *Last* represents whether an item was the last item a session clicked, and *n_clicks* tells how many times an item was clicked during a session. In Section 3.19 we found that these are important factors when deciding whether a buy-session ended up buying a specific item. *Prob* is the probability of a session buying the specific item - see Section 3.3.4. *Prob_click_dependent* is computed in a similar manner, but takes into account how many times the session has clicked the item. When calculating *prob_click_dependent*, we make one probability for when the items are

Feature	Type	Values	IG	IGR
last	Nominal	0 or 1	0.0423	0.0563
prob_click_dependent	Numeric	0 - 1	0.2250	0.0590
prob	Numeric	0 - 1	0.1408	0.0449
item_duration	Numeric	0 - ∞	0.0050	0.0091
n_clicks	Numeric	0 - ∞	0.0901	0.0782

Table 4.9: Iteration 2: Extracted features, items-bought

clicked once, and another probability for when they are clicked more than once. We observe that the *prob_click_dependent* has a much higher information gain and information gain ratio than *prob*. By calculating the probabilities in the manner we do for *prob_click_dependent* we are using information from the *n_clicks* feature. Thus, *n_clicks* and *prob_click_dependent* is heavily dependent, and *n_clicks* will not provide as much additional information to *prob_click_dependent* as to *prob*. We imagine that there are items that have a low probability of being bought, even when clicked twice, and that *prob_click_dependent* may prevent all items that are clicked twice being classified as buy.

4.4.2 Buy-or-not

When plotting the probability of a session ending up buying a certain item, given that the session had clicked the item, we found interesting results. We saw that some items had a much greater probability of being bought than others. We wanted to utilize this in our prediction of buy-sessions. There was extracted a feature related to the items a session has visited, *least_one_prob*. *Least_one_prob* is calculated by subtracting the probability of a session not buying any of the items it has visited from 1, that is the probability of a session buying at least one of the items. When computing *least_one_prob* we have assumed that each item is independent. For a session that have visited three items with the probabilities 0.4, 0.6 and 0.7 the *least_one_prob* would be $1-((1-0.4)\cdot(1-0.6)\cdot(1-0.7)) = 0.928$.

Feature	Type	Values	IG	IGR
avg_prob_clicks_dependent	Numeric	0.0-1.0	0.0255	0.00756
max_prob_clicks_dependent	Numeric	0.0-1.0	0.0344	0.0108
least_one_prob_clicks_dependent	Numeric	0.0-1.0	0.0422	0.01253
avg_prob	Numeric	0.0-1.0	0.0169	0.0057
max_prob	Numeric	0.0-1.0	0.0224	0.00732
least_one_prob	Numeric	0.0-1.0	0.0288	0.00949

Table 4.10: Iteration 2: Extracted features, buy-or-not

Further, we noticed that some items when visited led to a higher probability of a buy-session than others. This is affected by the probabilities of buying the different items. Visiting an item with a high probability of being bought increases the probability of a session buying something(e.g. that particular item). There were extracted two features related to this, *max_prob* and *avg_prob*. *Max_prob* is the maximum probability for a session of buying something when an item is clicked, and the *avg_prob* is the average probability for a session of buying something given all the items the session has clicked. Given the example session above the average probability would have been $\frac{0.4+0.6+0.7}{3} = 0.60$ and the maximum probability 0.70.

In addition *avg_prob*, *max_prob* and *least_one_prob* is also computed based on probabilities depending on the number of clicks - similar to the approach taken for items-bought. These are listed in Table 4.10, each ending with *clicks_dependent*.

4.4.3 Classification

We wanted to find out if the click-dependent probabilities gave better classifications than the non-click-dependent. First we tested for items-bought classification and afterwards for buy-or-not classification.

Items-bought

We tested with two different sets of features for items-bought classification. Set-1 consisted of *prob*, *first*, *last*, *item_duration* and *n_clicks* and Set-2 of *prob_click_dependent*, *first*, *last*, *item_duration* and *n_clicks*. The classification was performed using the Bayesian Network classifier in Weka.

Features	Precision	Recall	ROC	LS
Set-1	0.746	0.642	0.821	**8656**
Set-2	0.745	0.631	0.819	8472

Table 4.11: Iteration 2: Items-bought classification, BN

In Table 4.11 you can see that Set-1 performed slightly better than Set-2 on all evaluation measures. Even though *prob_click_dependent* gives more information than *prob* (Table 4.10), *n_clicks* and *prob* together seems to perform slightly better than *n_clicks* and *prob_click_dependent*.

Buy-or-not

When testing which probabilities that gave the best results for buy-or-not classification we tested with two different sets. One set consisting of click-dependent probabilities and another one with non-click-dependent probabilities. The resulting buy-sessions from the classification was further evaluated with the items-bought classifications found in the previous section. We used the same type of probabilities in each step (e.g when using click-dependent probabilities for buy-or-not we used click-dependent probabilities for items-bought too).

	Buy or not			Items bought			
Features	Precision	Recall	ROC	Precision	Recall	ROC	LS
Set-1	0.198	0.489	0.787	0.157	0.593	0.729	2477
Set-2	0.185	0.540	0.795	0.149	0.638	0.719	**2910**

Table 4.12: Iteration 2: End-to-end classification, BN for items-bought and BN for buy-or-not

In Table 4.12 we give the results from the buy-or-not testing. Set-1 is the set of features including non-click-dependent probabilities. From the Table 4.12 we see that end-to-end testing with click-dependent probabilities performs the best, with a score of 2951. For buy-or-not classification we observe that Set-1 gives higher precision, but a lower recall and ROC than Set-2. For items-bought classification the recall is a lot higher for click-dependent probability, but the precision and ROC is higher for non-click-dependent. Given the results from the previous section, where non-dependent-probability gave better results on all evaluation measures for items-bought classification, this is not as expected. Clearly it has do with the sessions the classifier gets as input. The click-dependent probability classification looks to be better at classifying the session-items from the sessions we have classified as buy - compared to the session-items from all buy-sessions.

	Buy or not			Items bought			
Features	Precision	Recall	ROC	Precision	Recall	ROC	LS
Set-2 B-or-N, Set-1 I-B	0.187	0.545	0.798	0.150	0.619	0.716	2852

Table 4.13: Iteration 2: End-to-end classification, BN for items-bought and BN for buy-or-not

To further investigate what probability that is the best for items-bought classification we ran another experiment. We used a Bayesian Network classifier built on dependent-probabilities to classify sessions as buy or not, and gave the resulting session-items to a Bayesian Network classifier built on non-click-dependent probability. In this way, we could compare the probabilities for items-bought classification on the same set of session-items. The result can be seen in Table 4.13. Still the end-to-end score is not as good as the end-to-end score with dependent probabilities. The precision for items-bought classification with click-dependent probability is slightly worse, while the recall is slightly better. The score tells us that the overall performance is slightly better when using click-dependent probability. However, since the difference in local score was so small we uploaded a solution to the RecSys Challenge for both approaches. The one using click-dependent probabilities for both buy-or-not and items-bought scored 0.3 percent better - with a score of 36248. The other approach scored 36143. Given the results we chose to continue using the click-dependent probabilities in the next iterations.

4.5 Iteration 3: Change of concept

So far we have predicted a consumer's behaviour by first looking at characteristics related to the session when classifying a session as a buy-session or not. Afterwards we have evaluated the items the session has visited to decide which items the session has bought. In this iteration we test a new approach, leaving out the first step and only predict if an item a session has visited will be bought or not. If we predict that an item x is bought by a session, this session will classified as a buy-session buying the item x - and possibly more items.

4.5.1 Classification

When performing classification of what items a session buys we used the same features as in the last iteration, see Table 4.9.

Bayesian Network was used in this iteration as well, as the only objective was to explore if it is necessary to split the problem into two sub tasks. This approach obtained a challenge score of 14326, which is a drastic decrease. This concept was therefore discarded.

4.6 Iteration 4: Handling unbalanced data set and testing algorithms for items-bought

In Section 2.2.5 we presented different approaches when it comes to handling unbalanced data sets. We wanted to test two of them, namely undersampling (us) and cost-sensitive (cs) learning. Oversampling is often helpful when the amount of data is low, which is assumed not the case here.

When performing undersampling we randomly removed $\left(1 - \frac{|\text{buy-sessions}|}{|\text{not-buy-sessions}|}\right) \cdot 100 = \left(1 - \frac{18327}{333310-18327}\right) \cdot 100 \approx 94.2$ percent of the not-buy-sessions from the training data. An overview of the resulting data set can be seen in Table 4.14.

Measure	Value
Number of clicks	177222
Number of sessions	36539
Number of sessions that buy	18327
Buy sessions/Sessions ratio	0.502
Buys per session	1.127

Table 4.14: Iteration 4: Overview training data, 50/50 buy-sessions and not-buy-sessions

We observe that the sessions in this new training set has about 50 percent probability of buying something, meaning buy-session and not-buy-sessions are approximately equally represented.

We will run items-bought classification with Random Forest, Bayesian Network, Decision Tree and Logistic Regression. Re-test of the algorithms for buy-or-not classification on an undersampled data set will also be performed. Undersampling will be compared to using cost-sensitive learning.

4.6.1 Classification

For items-bought prediction we will test the algorithms we tried for buy-or-not classification in Iteration 1, in addition to Random Forest, with the features we extracted in Iteration 2. As mentioned, we also test buy-or-not classification with undersampling and cost-sensitive learning. Naive Bayes was ignored in this iteration, but will be tested together with feature selection later on for both classification tasks.

Items-bought

The problems with unbalanced data set and dependencies between features are not as prominent for this step in the process as for buy-or-not classification. Thus we do not expect any of the algorithms to produce unreasonably poor results.

Classifier	Precision	Recall	ROC	LS
BN	0.745	0.631	0.819	8472
DT	0.643	0.646	0.673	8004
RF	0.688	0.686	0.796	8548
LR	0.727	0.701	0.8266	**8730**

Table 4.15: Iteration 4: Items-bought classification

As seen in Table 4.15, the Logistic Regression classifier performs best on items-bought classification, when looking at the scoring-function. It has the second best precision, only Bayesian Network is better, and a higher recall compared to all the other algorithms. Even though Logistic Regression performed better than Random Forest, the Random Forest algorithm may not be parametrized optimally (the default parametrization is used), which we think will improve its results. Optimization of the different algorithms and feature selection will be conducted in Iteration 6.

Buy or not

In the previous section we saw that Logistic Regression performed best on items-bought classification. When testing buy-or-not classification we therefore did end-to-end testing with Logistic Regression for items-bought classification. That is, the session-items belonging to sessions classified as buy-sessions were classified with Logistic Regression to produce local and challenge scores. When using cost-sensitive learning we used the ratio between buy-sessions and not-buy-sessions as penalization. Classifying a session as a false-negative is punished with 94.18, while classifying a session as a false-positive is punished with 5.82. The cost sensitive learner in Weka is used (see Section 2.2.5), with the parameters listed in Appendix B.

	Buy or not				Items bought				
Classifier	Tech	P	R	ROC	P	R	ROC	LS	CS
LR	us	0.146	0.677	0.795	0.128	0.720	0.681	3442	43830
RF	us	0.131	0.734	0.794	0.122	0.715	0.683	3469	**44288**
DT	us	0.097	0.655	0.652	0.105	0.710	0.700	2275	–
BN	us	0.137	0.732	0.797	0.122	0.707	0.713	**3474**	43930
LR	cs	0.147	0.665	0.720	0.128	0.721	0.681	3398	43066
RF	cs	0.133	0.688	0.713	0.125	0.712	0.694	3249	42061
DT	cs	0.163	0.449	0.652	0.160	0.725	0.684	2561	–
BN	cs	0.138	0.722	0.729	0.123	0.706	0.714	**3433**	**44103**

Table 4.16: Iteration 4: End-to-end classification, LR for items-bought

The differences between the cost-sensitive approach and undersampling are small. We get the best local score when using Random Forest (us). However, Logistic Regression (us), Bayesian Network (cs) and Logistic Regression (cs) scores almost as high. The Decision Tree stands out as the poorest classifier in both cases, but we see improvement from the first iteration. Because the differences in local scores are so small between the two approaches, we uploaded solutions to RecSys Challenge for the best performing algorithms. From Table 4.16 we see that the only classifier performing better with cost-sensitive learning is Bayesian Network. However, the difference in challenge score is only 0.3 percent. For Logistic Regression and Random Forest the differences are 1.8 and 5.3 percent respectively. Undersampling showed the overall best performance for challenge score, and will be used in the following iterations.

4.7 Iteration 5: New features and probabilities depending on time

The probabilities we have worked with up until now, are made up from the entire time period of the data set and based on whether a session has clicked once on an item or twice or more. Our analysis shows that the probabilities of buying an item varies dependent on what time the session occurs. This may vary from day to day, week to week, or month to month. In addition a time dependent probability is likely to capture variations caused by sales, which was not represented for all data in the data set. Based on this, we researched how incorporating time of occurrence into the probability features would affect the classification performance. The issue of making a probability for each item, for every day of the time period, is that the number of clicks and buys on a given day for a given item may be zero or very low. We have tried different regression techniques to capture the varying probability of buying an item - one of them being Loess Regression (see Appendix C). This approach proved to be too time consuming and did not provide sufficient results. Another approach taken was to use the probabilities from days. We looked at the probability of a session buying the item the day the click occurred, as well as probabilities for the surrounding days. We put a threshold on how many sessions that must have clicked the item before accepting the probability. We will call this feature *time_prob*. The first steps in this section was to find the best threshold for accepting the probabilities for *time_prob*. The click dependency researched in Iteration 4 is still incorporated in the probabilities.

Later in the section we will present two new features we extracted for items-bought classification and buy-or-not classification. These two features are related to a session clicking an item two or more times in a row.

4.7.1 Finding the best probability for items

We performed testing with different thresholds to improve the features built on *time_prob* - this was done for both items-bought classification and buy-or-not classification.

Items-bought

When searching for a threshold of number of clicks for accepting a given probability we tried 11 different levels: 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55. To take an example, a session has clicked an item 04.05.2014 and the threshold is set to 5. To find the probability of that session buying the item we first check if more than five other sessions have clicked the item the same day. If that is the case we set the probability to the number of sessions that has bought the item that day divided by the number of sessions that has clicked the item that day. Further, if the item is clicked 5 times or less we do not trust the probability for that specific day and continue searching by looking at the day after. If the accumulated number of sessions now are more than 5 we accept the probability, else we continue searching on the 03.05.2014. This process continues until the threshold is reached. If the threshold is not reached a weighted average between the probability found in the *time_prob* process and the average probability of buying an item is returned.

Thresholds	LS
Time_5	8943
Time_10	8992
Time_15	9024
Time_20	9026
Time_25	**9028**
Time_30	9020
Time_35	9026
Time_40	9017
Time_45	9015
Time_50	8996
Time_55	8993

Table 4.17: Iteration 5: Thresholds for time dependent probabilities, LR

In Table 4.17 we present the evaluation. To separate the different probabilities from each other we used local score. We observe that the performance of *time_prob* increases as the threshold increases, up to a certain level. This means that smoothed *time_prob* probabilities are preferred over very day-specific probabilities. When the threshold is set to 25 the local score reaches its peak. We observe that by using *time_prob* the performance of classification improves - compared to what we saw in the previous iteration.

Buy-or-not

Since we got a relatively low increase in performance by performing Loess Regression compared to using *time_prob*, and as the computational cost for finding good fractions were so high, we decided to not perform Loess Regression for the buy-or-not classification. In Table 4.18 you can see the results when testing different thresholds for *time_prob*. The end-to-end test for producing local scores was performed by using Logistic Regression (as this is a more stable classifier than RF), and with the threshold for *time_prob* for items-bought set to 25. As seen for items-bought, the higher threshold values score better than the lowest, and higher certainty is appreciated over day-specific probabilities. The local score reaches its peak at 4020 with the threshold set to 70. This is an improvement of what we saw in Iteration 4, and time_prob will be used in the further testing of buy-or-not classification. We decided not to output challenge scores for the thresholds. This mainly because of time consumption.

Threshold	LS
5	3671
10	3755
15	3827
20	3871
25	3910
30	3962
35	3980
40	3995
45	3993
50	3998
55	4009
60	4006
65	4011
70	**4020**
75	4014
80	4011
85	4014
90	4014
95	4010

Table 4.18: Iteration 5: Thresholds for time dependent probabilities, LR for items-bought and LR for buy-or-not

4.7.2 New features

In the following sections we introduce the new features extracted for both items-bought and buy-or-not classification.

Items-bought

Feature	Type	Values	IG	IGR
consecutive_returns	Numeric	0 - ∞	0.08665	0.1057
max_time_consecutive	Numeric	0 - ∞	0.09112	0.1054
time_prob_25	Numeric	0 - 1	0.25477	0.0664

Table 4.19: Iteration 5: Extracted features, items-bought

We extracted three new features for items-bought classification in this iteration. An overview can be seen in Table 4.19. *Consecutive_returns* and *max_time_consecutive* are related to a session clicking the same item twice or more in a row. *Consecutive_returns* is the maximum number of times this has happened for a session-item, while *max_time_consecutive* is the maximum duration between two such clicks. This behaviour is kind of strange and most likely related to the user-interface of the online-shop. We observed when analysing that this kind of behaviour dramatically increased the probability of the session ending up buying the item (see Figure A.5). This feature will be strongly correlated with the number of times a session has clicked an item, but it looked like it could be an even stronger signal for a buy. The last feature we extracted was *time_prob_25* - extracted as explained in Section 4.7.1. This feature will replace *prob_click_dependent* which was used in earlier iterations.

Buy-or-not

Feature	Type	Values	IG	IGR
max_consecutive_returns	Numeric	0 - ∞	0.0131	0.01229
max_time_consecutive	Numeric	0 - ∞	0.0136	0.0104
time_prob_least_one_70	Numeric	0 - 1	0.0495	0.01509
time_prob_max_70	Numeric	0 - 1	0.0412	0.01233
time_prob_avg_70	Numeric	0 - 1	0.0341	0.01025

Table 4.20: Iteration 5: Extracted features, buy-or-not

We extracted five new features. Three of these will replace features we already had in earlier stages of the process. *Time_prob_least_one_70* will replace *least_one_prob*, *Time_prob_max_70* will replace *max_prob* and *Time_prob_avg_70* will be used instead of *avg_prob*. The three new features related to the probabilities were extracted as explained in Section 4.7.1. Further, we extracted two features related to the behaviour of a session clicking the same item more than once in a row. *Max_consecutive_returns* tells the maximum number of times a session has clicked an item consecutively, while *max_time_consecutive* is the maximum duration between such clicks.

4.7.3 Classification

There have been extracted features for both items-bought classification and buy-or-not classification. We wanted to know how these features affected the performance of the classification. At least we expect the changes made to the probability features to increase the performance. When testing for items-bought we used Logistic Regression, as this classifier performed best in the last iteration. By the same argument we test buy-or-not classification with Random Forest.

Items-bought

We tested items-bought classification with two sets of features. Set-1 contained the features we had extracted before this iteration started, but with *click_dependent_prob* replaced with *time_prob_25*. The second set contained the same features as in Set-1 as well as *max_time_consecutive* and *consecutive_returns*.

Features	Precision	Recall	ROC	LS
Set-1	0.751	0.731	0.847	9028
Set-2	0.750	0.732	0.847	**9030**

Table 4.21: Iteration 5: Items-bought classification, LR

As expected, we observe from the results in Table 4.21 that the new probability feature makes the Logistic Regression perform better than in earlier iterations. When using the old probability in Iteration 4 we got a score of 8730. Further, we notice that the two new features we included related to a session clicking an item more than once in a row does not affect the performance drastically.

Buy-or-not

As for items-bought classification we chose to test two different set of features, to better see the influence of the features we had extracted. Set-1 consists of all the features we had extracted in Iteration 4, with the click-dependent probabilities

	Buy or not			Items bought			
Features	Precision	Recall	ROC	Precision	Recall	ROC	LS
Set-1	0.139	0.752	0.813	0.134	0.741	0.699	3972
Set-2	0.140	0.757	0.815	0.135	0.741	0.699	**4021**

Table 4.22: Iteration 5: End-to-end classification, LR for items-bought and RF for buy-or-not

replaced with *time_prob* probabilities. In Set-2 the features related to consecutive clicks on an item is also included.

In the previous iteration we got a local score of 3469, precision at 0.131 and recall at 0.734 for buy-or-not classification, using Random Forest. In Table 4.22 we observe that replacing the old probabilities with new ones has increased the performance on all evaluation measures. We also observe that adding the features related to consecutive returns slightly increased the recall for buy-or-not classification. Moreover, the score has increased further by close to one percent. When we uploaded a solution to the RecSys challenge we got a score of 49957. We included all the features we extracted in this iteration into the next. In that iteration we will re-run a set of the classification algorithms we have used so far, with and without feature selection, as well as optimizing parameters for Random Forest and Logistic Regression. The Decision Tree will not be included as it has produced poor results compared to the other algorithms.

4.8 Iteration 6: Optimizing algorithms

We have seen that both Logistic Regression and Random Forest have performed well after solving the issue of unbalanced data by undersampling. As both these algorithms have several possible parameters that can tune the classification to better fit data sets with specific characteristics, we evaluated both classifiers again, with the latest set of features. In addition to finding the best parameters, we also performed feature selection for a set of classifiers, by using an implementation in Weka that optimizes the set of features with regards to one specific classifier (see Section 2.2.4). The parameter search can be seen in Section 4.8.1, and feature selection in Section 4.8.2

4.8.1 Parameters

The first step was to find the parameters we wanted to test. This was performed by using a grid search implemented in scikit-learn (see Section 2.5). One aspect in terms of parameters is how well the classifier should be fitted to the training data, in order to find a middle ground between overfitting and underfitting. Further the two classifiers have different possible parameters, which thus were looked at separately. The measure used to select the best parameters is ROC. ROC was used as this measure evaluates the algorithms on different discrimination levels and is an established measure for unbalanced data sets (see Section 2.4.2). Optimally this should have been done using the RecSys score, which would have required a tailored implementation.

Random Forest

Two of the parameters we evaluate for Random Forest are concerned with the concept of how fitted the model should be to the training data. The first one is a parameter called *min_samples_split*, which gives a minimum threshold of how many samples that have to be present within an internal node in a tree in order to split it into new nodes. A lower number of allowed samples will result in a more tightly fitted classifier, while a higher number will yield a more general classifier with the risk of information loss. The second parameter concerning this is called *max_depth* and sets a number indicating how deep a tree is allowed to be built. A deeper tree will have more node splits and thus a more tightly fitted tree. If this parameter is set to *None*, the tree will build until all leaf nodes are only contain samples of the same class label, or until all leaf nodes contain less samples than the value set in the parameter *min_samples_split*. The optimal value of both these parameters will vary dependent on the number of training samples. The third parameter we tested was the evaluation measure for deciding which feature should be used for splitting a node. We used Entropy and Gini, which are two widely used split criteria [30]. For a single decision tree the default setting is to consider all features in the training data before splitting a node. Random Forest builds several decision trees, and uses a vote from each tree for making classifications. In order to build trees more robust to noise and outliers, it can be an advantage to not use all features to build all trees (see Section 2.3.4). The Random Forest implementation thus offer a parameter *max_features* where the number of features to consider at each split are set.

Parameter	Values
max_depth	3, 7, None
min_samples_split	1, 50, 100, 150, 200, 250, 300, 350, 400
criterion	gini, entropy
max_features	1, 7, auto

Table 4.23: Iteration 6: Parameter grid for Random Forest

Logistic Regression

The variable concerning overfitting/underfitting for Logistic Regression is named C, which sets the regularization strength, in that a higher value of C will yield a more tightly fitted classifier. Another important aspect for Logistic Regression is how to optimize the weights used by the model. We tested three different methods for this optimization: *newton-cg*, *lbfgs* and *liblinear*. As loss function, L_1 and L_2 were tested (see Section 2.3.5), which use absolute sum of error and sum of square respectively as inputs to the optimization method. Only *liblinear* supports the L_1 function. The final parameter we tested is called *max_iter* and is mainly important in order to control the computational time, by giving a maximum number of iterations used before the optimization with *newton-cg* and *lbfgs* is considered converged.

Parameter	Values
C	0.01, 0.1, 1.0, 10, 100, 1000
solver	newton_cg, lbfgs, liblinear
penalty	L1, L2
max_iter	100, 300, 500

Table 4.24: Iteration 6: Parameter grid for Logistic Regression

The results from parameter grid search for Random Forest and Logistic Regression can be seen in the list below.

- Items bought
  - Random Forest
    - max_features = auto
    - min_samples_split = 350
    - criterion = entropy
    - max_depth = None
  - Logistic Regression
    - penalty = l2
    - C = 1000
    - max_iter = 100
    - solver = liblinear
- Buy or not
  - Random Forest

  - max_features = 7
  - min_samples_split = 200
  - criterion = entropy
  - max_depth = None

- Logistic Regression
  - penalty = l2
  - C = 1000
  - max_iter = 300
  - solver = newton-cg

4.8.2 Feature Selection

As stated in the introduction of Iteration 6, we performed feature selection. The method for feature selection implemented by Weka can be read about in Section 2.2.4. We used ROC as evaluation measure here as well, and the rest of the parameters can be seen in Appendix B. Table 4.25 displays the best features for items-bought classification.

Classifier	Features
Random Forest	first, last, time_prob
Logistic Regression	first, last, item_duration, time_prob, n_clicks, max_time_consecutive, consecutive_returns
Naive Bayes	firs, last, time_prob
Bayesian Network	first, last, item_duration, time_prob

Table 4.25: Iteration 6: Features after running feature selection, items-bought

Further we also performed feature selection for buy-or-not classification, of which the results are listed in Table 4.26.

4.8.3 Classification

In this section we will evaluate the different algorithms with optimized parameters, with and without feature selection.

Items-bought

The Naive Bayes classifier has not been evaluated since Iteration 1, where it produced overall good results, and will be re-evaluated in this iteration. In addition we have tested Logistic Regression, Bayesian Network and Random Forest. All

Classifier	Features
Random Forest	max_time_consecutive, time_prob_avg, time_prob_max, avg_time_perclick, duration, time_prob_least_one
Logistic Regression	max_return, max_time_consecutive, time_prob_avg, clicks, avg_clicks, avg_time_perclick, time_prob_least_one
Naive Bayes	consecutive_returns, time_prob_avg, time_prob_max, avg_time_perclick, time_prob_least_one
Bayesian Network	max_return, time_prob_avg, max_time_between_clicks, avg_time_perclick, time_prob_least_one

Table 4.26: Iteration 6: Features after running feature selection, buy-or-not

algorithms were tested with and without feature selection, and Random Forest and Logistic Regression were run with the parameters found in Section 4.8.1.

	With Feature Selection				Without Feature Selection			
Classifier	P	R	ROC	LS	P	R	ROC	LS
RF	0.740	0.740	0.841	9059	0.750	0.750	0.853	**9272**
LR	0.745	0.732	0.845	8960	0.745	0.732	0.845	8960
NB	0.739	0.740	0.844	8976	0.826	0.386	0.835	6047
BN	0.752	0.721	0.843	9112	0.777	0.579	0.831	8028

Table 4.27: Iteration 6: Items-bought classification, with/without feature selection

As seen in Table 4.27, Naive Bayes performed a lot better after feature selection, which indicate that the full feature set contains dependencies. Naive Bayes assumes conditional independence between features given the class. Thus, the probability estimates outputted from the classifier when this assumption is not met will be lower than their real probabilities. Thus we observe that the precision decreases, while the recall increases. Bayesian Network also improves after feature selection, which is rather surprising as a Bayesian Network can work as a feature selector at its own (Hruschka et al. [17]). Further, we observe that the performance of the Random Forest algorithm has decreased with regards to ROC after feature selection. This is rather strange as the the wrapper should have maximized the ROC-score of the classifiers.

Buy-or-not

	Buy-or-not			Items-bought			
Classifier	P	R	ROC	P	R	ROC	LS
RF	0.148	0.748	0.822	0.139	0.755	0.715	**4178**
LR	0.160	0.686	0.816	0.148	0.755	0.710	4007
NB	0.171	0.547	0.791	0.165	0.754	0.742	3362
BN	0.146	0.735	0.812	0.136	0.746	0.735	4005

Table 4.28: Iteration 6: End-to-end classification, without feature selection. Items-bought with RF

	Buy-or-not			Items-bought			
Classifier	P	R	ROC	P	R	ROC	LS
RF	0.148	0.750	0.822	0.139	0.750	0.717	**4149**
LR	0.158	0.685	0.815	0.147	0.762	0.711	4015
NB	0.169	0.594	0.800	0.163	0.755	0.723	3661
BN	0.143	0.769	0.821	0.133	0.754	0.725	4129

Table 4.29: Iteration 6: End-to-end classification, with feature selection. Items-bought with RF

In Table 4.29 we observe that Naive Bayes, Bayesian Network and Logistic Regression perform better after feature selection, when it comes to local score. Again, this is as expected for Naive Bayes, but rather surprising for Bayesian Network. Random Forest performs best, and performs more or less similar with and without feature selection.

	Buy or not			Items bought				
Feature selection	P	R	ROC	P	R	ROC	LS	CS
no	0.148	0.748	0.822	0.139	0.755	0.715	**4178**	**52670**
yes	0.148	0.750	0.822	0.139	0.750	0.717	4149	52635

Table 4.30: Iteration 6: End-to-end classification, RF for both items-bought and RF for buy-or-not

Since the difference in local score was so small for Random Forest we ran an end-to-end test with challenge score. The results are seen in Table 4.30. In conclusion to this iteration we ended up with Random Forest without feature selection as the best classifier for both buy-or-not classification and items-bought classification, after finding the optimal parameters.

4.9 Iteration 7: Using multiple models

The data from the RecSys Challenge spans over several months. It is not unlikely that the user behaviour varies over time, and that a signal for a buy is not the same in April as in August. Change of consumer behaviour may be related to the time of year or changes made to the online store. In this section we make several models, one for each time period. The time periods should be as small as possible, but one still has to have enough data to build reliable classification models. There will be experimented with a time period set to one day, one week, two weeks and one month - and all the available data will be used. We will use Random Forest with the optimal parameters found in the previous section for all models, although this may not be optimal for all periods - because of the varying amount of data. It would have been too time consuming to optimize the parameters towards all time period lengths, although this could have improved the results. The only change made to items-bought classification is the number of classification models, thus the section about items-bought will be left out.

4.9.1 Buy-or-not

In Iteration 5 we introduced undersampling of the data set for buy-or-not classification. We undersampled the data set by using the ratio between buy-sessions and not-buy-sessions for the entire time period. When we are to make classification models for different time periods we have to use the ratio in the given time period when undersampling. Not-buy-sessions are undersampled by randomly removing $\left(1 - \frac{|\text{buy-sessions}|}{|\text{no-buy-sessions}|}\right) \cdot 100$ percent of the sessions.

4.9.2 Classification

When the time period was set to month, we used the six months represented in the data set, while when using other time periods we iteratively picked the respective number of days from the start day. In Table 4.31 the results for each month, when using one month as time period, are shown. The comparison between the different time periods is shown in Table 4.32.
We observe that the results for the first month are drastically better than the results for the other months, which are more similar to each other. One reason for why we observe this could be noise in the test set we are using, but we can not exclude the possibility that it actually is easier to classify consumer behaviour in the first month. Further, we see that the local score has increased from 4178, in Iteration 6, to 4413. The large increase in local score is most likely caused by the differences between the first month and the others. To further explore the differences between the months we uploaded a solution to the RecSys Challenge.

	Buy or not			Items bought		
Month	P	R	ROC	P	R	ROC
Month 1	0.205	0.866	0.889	0.230	0.874	0.786
Month 2	0.141	0.746	0.815	0.126	0.748	0.722
Month 3	0.137	0.712	0.801	0.125	0.727	0.705
Month 4	0.143	0.728	0.804	0.132	0.726	0.687
Month 5	0.148	0.750	0.832	0.143	0.759	0.702
Month 6	0.145	0.752	0.830	0.135	0.710	0.682

Table 4.31: Iteration 7: End-to-end classification, RF for items-bought and RF buy-or-not. Comparison of months

If the increase in challenge score showed similar behaviour as the local score, we could with a higher certainty say that the strange behaviour we observed locally was not a result of noise in the test set.

	Buy or not			Items bought			LS	CS
Month	0.153	0.762	0.833	0.145	0.758	0.721	4413	**55349**
Two weeks	0.153	0.762	0.834	0.145	0.759	0.722	4424	55272
One week	0.153	0.764	0.8235	0.144	0.760	0.726	**4437**	55135
One day	0.144	0.748	0.823	0.137	0.762	0.731	4184	-

Table 4.32: Iteration 7: End-to-end classification, RF for items-bought and RF buy-or-not. Comparison of time periods

In the first line in Table 4.32 the challenge score when using months as time period is shown. Iteration 6 ended up with a challenge score of 52670, and it has now increased to 55349. Thus the peculiar observations from the test data seems to be represented in the challenge data as well. Anyways this shows that there are differences in terms of behavior between the months. After researching this phenomenon further, we saw that almost all items bought in the first month were clicked twice. This was not the case in later months. The results for two weeks, one week and one day can also be seen in Table 4.32. It can look like the amount of data is not sufficient when the time period is set to one day, as it performs worse than the rest. The results for one month, two weeks and one week are rather similar, and making the period shorter than one month does not affect the performance drastically. This could be because the largest difference is found between the first month and the rest of the period, and models based on months will also capture this.

4.10 Iteration 8: Optimizing towards scoring-function

In this iteration we look closer at the scoring-function the RecSys Challenge uses for evaluating solutions. The scoring-function is as follows:

$$Score(\mathbf{Sl}) = \sum_{\forall s \in \mathbf{Sl}} \begin{cases} \frac{|S_b|}{|S|} + \frac{|A_s \cap B_s|}{|A_s \cup B_s|} & \text{if s} \in S_b \\ -\frac{|S_b|}{|S|} & \text{else} \end{cases} \tag{4.3}$$

where **Sl** is the sessions you have predicted buys for; S_b is the sessions that actually have bought something; S is all the sessions; A_s is the set of items that the session has bought; and B_s is the set of items that are predicted as bought by the session. Let $P(Buy) = p$, the expected score of predicting a session as a buy session is then:

$$p \cdot \left(\frac{|S_b|}{|S|} + \frac{|A_s \cap B_s|}{|A_s \cup B_s|} \right) + (1-p) \cdot \left(\frac{-|S_b|}{|S|} \right) \tag{4.4}$$

We observe that most of the variables in the equation above are known. The only terms unknown is p and $\frac{|A \cap B|}{|A \cup B|}$, and thus the only terms that need to be estimated. In the next sections we will look at how items-bought and buy-or-not classification can be used for solving these tasks.

4.10.1 Buy-or-not optimization

Most classifiers can produce probability estimations of an instance belonging to a given class. Some classifiers do this naturally, as Naive Bayes and Logistic Regression, while others use techniques related to their structure. An example is the Decision Tree which can use the fraction of positive and negative samples in a leaf node to produce a probability estimate for an unclassified sample. In this iteration we make use of the probability estimates the classifiers output to estimate the probability of a session buying, that is $P(buy)$.

When exploring what classifier that produces the best probabilities we have used Brier score (see Section 2.4.2). The Brier score measures accuracy for probabilistic predictions. From Table 4.33 we observe that Random Forest is the classifier performing best when it comes to Brier score, and thus outputs the best probability estimates. In Figure 4.1 you can see a plot of how well the probability estimates are. We have plotted the probability estimates from Random Forest for the second month in the time period versus a perfectly calibrated classifier. There are methods for improving probability estimates (Niculescu-mizil and Caruana [28]). However, the estimates outputted from Random Forest already follows the calibration curve well.

Since we have trained our classifiers on an undersampled training set the probability estimates from the classifiers do not reflect the real probability of a

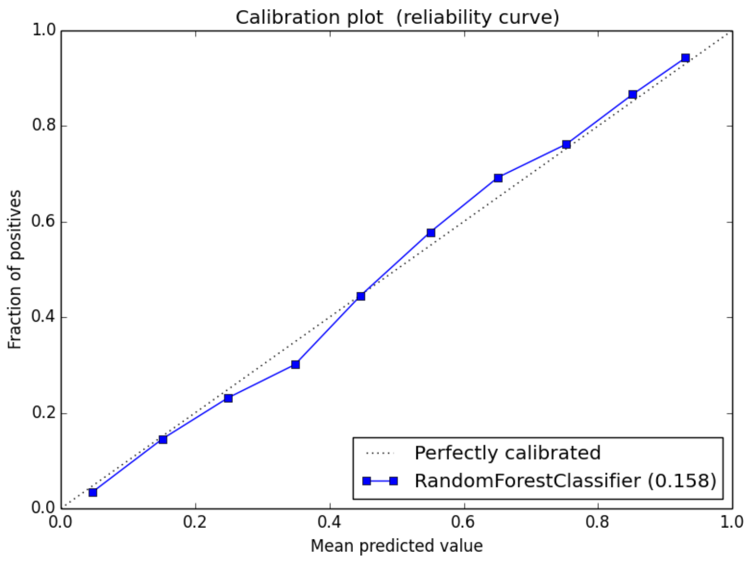


Figure 4.1: Calibration plot, buy-or-not for second month using RF vs perfectly calibrated classifier (code adapted from [26])

Classifier	Calibration	Brier score
Random Forest	None	0.158
Bayesian Network	None	0.199
Naive Bayes	None	0.230
Logistic Regression	None	0.230

Table 4.33: Iteration 8: Brier score for different algorithms, buy-or-not

session ending up buying. A model trained on an undersampled set will produce higher probabilities for the positive class than a classifier trained on the unmodified data set. We trained our model on a data set where $\text{ratio}_{\text{undersampled}} = \frac{1}{1}$ between the positive and the negative samples, and $\text{ratio}_{\text{real}} = \frac{|\text{not-buy-sessions}|}{|\text{buy-sessions}|}$. Thus we have to rescale the probability estimates for them to make sense. Let x be the probability outputted from a classifier trained on the undersampled training set for a given test sample. The classifier thus states that $x \cdot 100$ out of hundred such samples are positive, and $(1-x) \cdot 100$ are negative. Given that the data set was undersampled, the actual number of negative samples is estimated to $(1-x) \cdot 100 \cdot \text{ratio}_{\text{real}}$. Thus we scale the probabilities as follows:

$$p_{new} = \frac{p_{old}}{p_{old} + (1 - p_{old}) \cdot \frac{|\text{not-buy-sessions}|}{|\text{buy-sessions}|}} \tag{4.5}$$

where p_{new} is the probability we will use in our optimization and p_{old} is the probability from the classifier trained on the undersampled training set. For $\frac{|\text{not-buy-sessions}|}{|\text{buy-session}|}$ we will use the ratio between buys and not-buys from the hour the session was active, and thus incorporating information about the probability of a session buying something given the exact time. As we saw in Figure 3.1, Figure 3.2 and Figure 3.4 the probability of a session buying varies over time.

4.10.2 Items-bought optimization

We estimated the probability for a session ending up buying, the next step is to estimate the term $\frac{|A \cap B|}{|A \cup B|}$. Up to this point classification has been used for deciding whether the session ended up buying an item or not. Now, as for buy-or-not, we make use of the probabilities the classifiers output. These probabilities can be used for computing the maximum expected score for the term $\frac{|A_s \cap B_s|}{|A_s \cup B_s|}$ per session.

To introduce the approach for finding the maximum expected score we will look at an example where the session has visited only two items, it_1 and it_2. Further, we will assume that the classifier has estimated that the probability of the session buying it_1 is larger than it_2, $P(it_1) > P(it_2)$. Given that the session has visited two items we face three interesting choices for what we could predict. The session can buy either it_1 or it_2, or it can buy both. We will now derive equations for the maximum expected score for the three different cases. Let p_1 be the probability estimate of the session buying it_1, and let p_2 be the same for the second item. The maximum expected scores are then

$$Exp(\text{buy } it_1 \text{ and } \neg\text{buy } it_2) = \tfrac{p_1 \cdot p_2}{2} + p_1 \cdot (1-p_2) \cdot 1 + p_2(1-p_1) \cdot 0 = p_1 - \tfrac{p_1 \cdot p_2}{2} \tag{4.6}$$

$$Exp(\neg\text{buy } it_1 \text{ and buy } it_2) = \tfrac{p_1 \cdot p_2}{2} + p_1 \cdot (1-p_2) \cdot 0 + p_2(1-p_1) \cdot 1 = p_2 - \tfrac{p_1 \cdot p_2}{2} \tag{4.7}$$

$$Exp(\text{buy } it_1 \text{ and buy } it_2) = p_1 \cdot p_2 \cdot 1 + \tfrac{p_1 \cdot (1-p_2)}{2} + \tfrac{p_2 \cdot (1-p_1)}{2} = \tfrac{p_1}{2} + \tfrac{p_2}{2} \tag{4.8}$$

We see from the equations that predicting that the session buys only it_2 never is better than predicting only it_1, given $p_1 > p_2$. Thus, one will never predict an item as bought unless all the items with higher probability also are predicted as bought. We also want to know when we should predict that the session bought both items instead of only the item with the highest probability, and opposite. Say both if

$$Exp(\text{buy } it_1 \text{ and buy } it_2) > Exp(\text{buy } it_1 \text{ and } \neg\text{buy } it_2) \implies p_1 < \tfrac{p_2}{1-p_2} \tag{4.9}$$

and say only it_1 if

$$Exp(\text{buy } it_1 \text{ and buy } it_2) \leq Exp(\text{buy } it_1 \text{ and } \neg\text{buy } it_2) \implies p_1 \geq \tfrac{p_2}{1-p_1} \tag{4.10}$$

In Figure 4.2 you can see a graphical representation of the probability thresholds for two items.

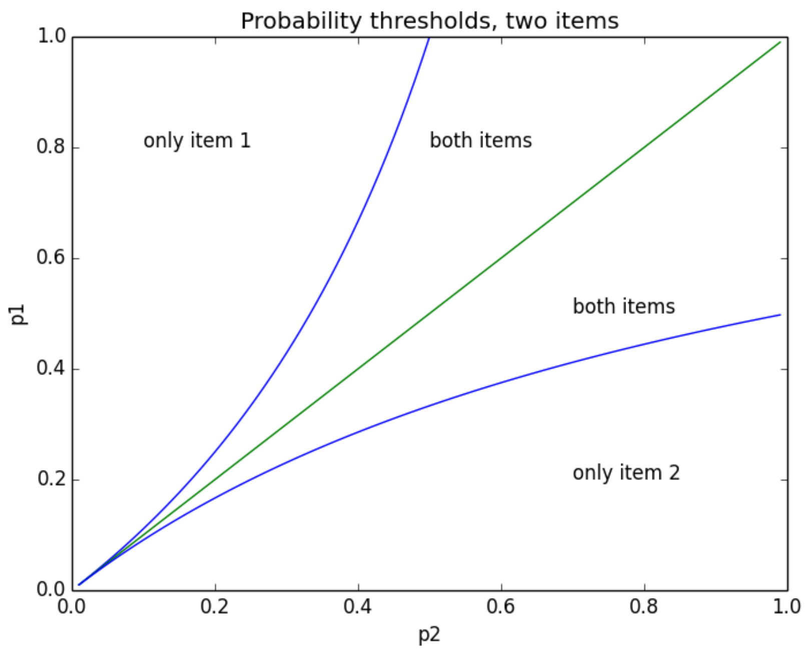


Figure 4.2: Probability thresholds for two items

Generally finding the maximum expected score is done by sorting the items a session has visited by the probability estimates from the classifier. After this is done you compute the expected score like explained above for: buying the first item; then for buying the first and second item; and so on. The maximum of these computations is set to be the maximum expected score.

$item_1$	$item_2$	$item_3$
0.7	0.4	0.1

Table 4.34: Iteration 8: Probability for buying items, example

We will now go through an example where a session has clicked three different items. The probabilities of the session buying the different items are presented in Table 4.34. We observe that the items are sorted. The next step is to compute the probability of each combination of items. We will compute the probability of the session buying no items first, and then continue in a binary counting order until we have reached 111, where all items are bought. This is illustrated in Table 4.35

$item_1$	$item_2$	$item_3$	probability
0	0	0	0.162
0	0	1	0.018
0	1	0	0.108
0	1	1	0.012
1	0	0	0.378
1	0	1	0.042
1	1	0	0.252
1	1	1	0.028

Table 4.35: Iteration 8: Probability of all item combinations, example

Finally, we have to compute the maximum expected score. As mentioned above we have to compute the expected score for three different cases. We will start with the one where the session only buys $item_1$. We observe in 4.35 that there are four cases where the session ends up buying $item_1$. The expected score is computed as follows

$$\begin{aligned} E_{score}(1,0,0) &= P([1,0,0]) \cdot \frac{[1,0,0] \cdot [1,0,0]}{[1,0,0] \vee [1,0,0]} + ... + P([1,1,1]) \cdot \frac{[1,0,0] \cdot [1,1,1]}{[1,0,0] \vee [1,1,1]} \\ &= 0.378 \cdot \frac{1}{1} + 0.042 \cdot \frac{1}{2} + 0.252 \cdot \frac{1}{2} + 0.028 \cdot \frac{1}{3} = 0.534 \end{aligned} \tag{4.11}$$

For the other two cases the scores are respectively 0.54([1,1,0]) and 0.40([1,1,1]).

The maximum expected value is thus 0.54, and we should predict that the session ends up buying $item_1$ and $item_2$ to maximize our score.

Classifier	Calibration	Brier score
Random Forest	None	0.156
Bayesian Network	None	0.194
Naive Bayes	None	0.173
Logistic Regression	None	0.160

Table 4.36: Iteration 8: Brier score for different algorithms, items-bought

As for buy-or-not classification we have evaluated a set of classifiers with respect to their probability estimates by using Brier score. In Table 4.36 it is shown that the Random Forest classifier outputs the best probability estimates. As for buy-or-not we have drawn the probability estimates from the Random Forest classifier against a perfectly calibrated classifier. The result can be seen in Figure 4.3. As for buy-or-not the probability estimates from Random Forest follows the perfectly calibrated line well.

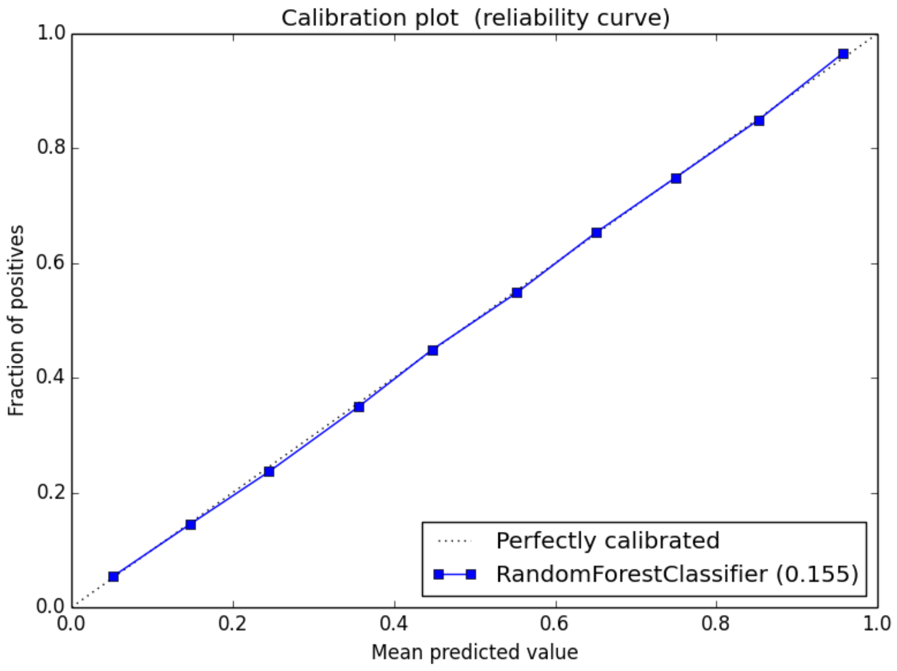


Figure 4.3: Calibration plot, items-bought for second month using RF vs perfectly calibrated classifier (code adapted from [26])

4.10.3 Classification

When doing classification we have set the time period to month, two weeks, one week and one day. Random Forest is used for both items-bought and buy-or-not classification, with the optimized parameters from Iteration 6, as this classifier gave the best probability estimates. A session is said to be a buy-session with the items providing the maximum expected score if and only if:

$$p \cdot \left(\frac{|S_b|}{|S|} + \frac{|A_s \cap B_s|}{|A_s \cup B_s|} \right) + (1-p) \cdot \left(\frac{-|S_b|}{|S|} \right) > 0 \qquad (4.12)$$

In Table 4.37 we present the final results we got in the RecSys challenge. We observe that the time periods performed more or less similar, except when the time period was set to one day. However, the results when using one day is closer to the others in performance of local score now than in the previous iteration. The probability estimates from the classifier when using one day looks

Time period	LS	CS
Month	4624	56778.7
Two weeks	4650	**56944.6**
Week	4651	56852.5
One day	4592	-

Table 4.37: Iteration 8: Final results in the RecSys Challenge

to be almost as accurate. We get the best performance when the time period is set to two weeks. In Iteration 7 we stated that we wanted to find a time period as small as possible, which still provided enough data to build reliable classification models. A time period that spans over two weeks thus looks to be the middle ground between these two concerns.

Chapter 5

Evaluation and Conclusion

In Section 5.1 we evaluate the research objectives presented in Section 1.2. In Section 5.2 we consider how recommender systems can apply our findings to increase revenue and user satisfaction, and we summarize our contributions in Section 5.3. The last section comprises of suggestions of how to further develop the concepts of classifying sessions in e-commerce.

5.1 Evaluation

In Section 1.2 we stated two research objectives. The first research objective was to decide if a session from an online store ends up buying or not. Given our data set, guessing randomly if a session bought something or not would have yielded approximately a precision and recall at 0.055 and 0.50 respectively. We have shown by using classification that this performance can be drastically improved. In Section 4.9 the precision and recall had increased to 0.153 and 0.762 respectively. Through the iterations in Chapter 4 the probabilities of buying the respective items proved to be the largest contributor towards the results. Some items have a higher probability of being bought or influencing a session to buy, when clicked. Moreover, it is important to take the time of occurrence into consideration. An item's probability of being bought varies over days, and further also depends heavily on the time of the day. The variance between days is most likely affected by when the items are on sale and what season it is. Consumers are more likely to buy a lawn mower at the start of the summer than at the end, and the probability of buying an item may increase when the item is on sale. Even more intuitive is it that sessions with a longer duration and a higher number of clicks have a higher probability of ending up buying. Further, several clicks on same item during a session is a strong buy-signal for the session - e.g buying that particular item. However, this does not hold for all items. There exist certain items that still have a low probability of leading towards buys, or ending up being bought, even when clicked twice or more. We observed this when making the probabilities for buying a specific item dependent on the number of clicks the session had on that particular item.

The second objective was to decide what items a predicted buy-session bought. By randomly guessing what items a session - from a set consisting of sessions produced by our buy-or-not classification in Section 4.9 - bought we got a precision, recall and score of 0.085, 0.421 and 20697 respectively. By using Random Forest for classification we improved the precision, recall and score to 0.145, 0.758 and 55349 respectively. Repeatedly clicking an item turned out to a be a strong signal for the session ending up buying that item. We also explored the differences between items when it comes to how likely it is that a buy-session ends up buying it, given that the buy-session has clicked it. The difference from the probabilities used for buy-or-not classification is that only actions made by buy-sessions are considered. Moreover, as for buy-or-not classification, we introduced the time of occurrence and number of clicks into the probabilities. The differences between items proved to be the most important factor when deciding what items a session purchased. Further, the interest a session has for an item is also associated with the time of interaction during the session. A session often has a high interest in the first item it clicks, as well as the last.

We also observed that splitting the training data in several time periods and making a classification model for each of them - instead of having one global model - increased the performance for both buy-or-not classification and items-bought classification. One reason for this was that the behaviour in the first month drastically stood out from the rest, making the global model too general. What we observed was that almost all items bought where clicked two times or more when bought in the first month, making this a really strong buy-signal. This might be caused by the design of the e-commerce site, which may have been changed after the first month of the period - introducing new behavioral indicators of buy. Thus, it can be important to revise how a consumer's actions should be interpreted after doing modifications to the design and functionality of an e-commerce site.

After the final step - where we used the probabilities outputted from the classifier to optimize the solution towards the RecSys scoring function - we got a final score of 56944. This gave us a top 5 ranking, among 338 teams, in the RecSys Challenge one week before the deadline.

The problems of deciding if a session ended up buying and what it bought have been looked at as more or less two separate tasks. It could have been more beneficial to look at these tasks as more dependent of each other. How to predict what items a session buys is most likely dependent on what sessions you are to predict buys for. Our approach for optimizing the solution can be seen in 4.2.3 - where a more dependent, iterative alternative also is presented. The amount of training and test data used is small compared to the available amount of data - at least for the first iterations. It can be argued that larger data sets would have yielded more accurate results, and made the decisions towards the final solution more optimal. It should have been tested more thoroughly if the amount of training and test data was sufficient.

5.2 Discussion

We have extracted information from anonymous, sparse session data from an e-commerce site and used this information to predict buys of consumers. In the following we discuss other applications for the information extracted.

In practice, recommender systems applies information about users and their previous interactions, in order to make suitable suggestions. This information is based on either implicit or explicit ratings from the users. Explicit ratings are produced by letting the user give feedback for an item, while implicit ratings are calculated by the system itself based on the user's interactions with an item. Recommender systems usually make use of several types of events - such as clicks, adding to chart, current site view etc. - when calculating implicit ratings. We do not possess information about other events than items clicked. However, the information we have extracted contains indicators of user preferences, which may be valuable when making implicit ratings for a recommender system.

Most recommender systems gather large amounts of data about users, which creates a risk of leakage of private information. As stated by Jeckmans et al. [18], many users do not know how much and if their data is collected, how securely this information is stored, and if the information is sold to third parties. McSherry and Mironov [25] look at an example of how a person can take advantage of a recommender system to discover the interests of users and reveal their identity. In the example the anonymous data set from the Netflix Prize - a competition held by Netflix on making a movie recommender system [27] - was used. A person X was able to link user Y's stored Netflix Prize Dataset record to Y's public IMDB profile, thus matching an anonymous data record to a public profile. Although movies and movie recommendations might not be the most private matter, this example illustrates one of the pitfalls of storing information about consumers. Given this, one can imagine that there is an increasing number of consumers that wish to be anonymous when using recommendation systems - e.g using incognito mode in the web browser, where the web browser does not remember the browsing history. Thus recommender systems need to be able to handle recommendations based on anonymous session data. By using classification we have extracted information solely based on such data, which can be of use in these situations. Moreover, one could imagine that recommender systems solely based on session data will emerge in the future - providing the desire of privacy for the users.

5.3 Contributions

We have shown that one can gather information about an online e-commerce user's preference solely based on anonymous sparse session data. This information has been used for predicting if a session bought, and in that case, what items it bought. Given the concerns regarding privacy and leakage of information, recommender systems face a challenge in the future of being able to cope with anonymous data. The way to extract information from session data used in this thesis may be of interest for this research field. The information provided can also be useful when making implicit ratings that are to be used in a recommender system.

5.4 Future Work

When making decisions about which items a session bought we have assumed that the items are independent of each other. More formally we have assumed that $P(\text{buy } it_1 \wedge \text{buy } it_2) = P(\text{buy } it_1) \cdot P(\text{buy } it_2)$, where it_1 and it_2 are two items (see Section 4.10). Such an assumption may not be correct - the probability of buying it_2 may be dependent of which item it_1 is. We have tried several strategies for solving this problem, but none of these has shown good results. We think that there is potential improvement if one can find a good way of finding the dependencies between items.

Appendix A

Analysis

A.1 Buy-or-not

Here we present the rest of the analysis conducted for buy-or-not classification.

A.1.1 Time

If a session watches an item for a short amount of time, this could mean that the session is more like a surfer (only browsing the website) than a buyer. Moreover, if a session spends some time watching all the items it has visited we imagine that this session closely considers which of the items it likes the best, and is therefore a more probable buyer. In Figure A.1 we see that the probability of a session ends up buying something increases as the minimum time spent watching an item increases. If a session has watched an item for close to 0 seconds the probability of this session buying is close to 1 percent. If a session has watched all his items for above 100 seconds the chances of that session buying is close to 5 percent.

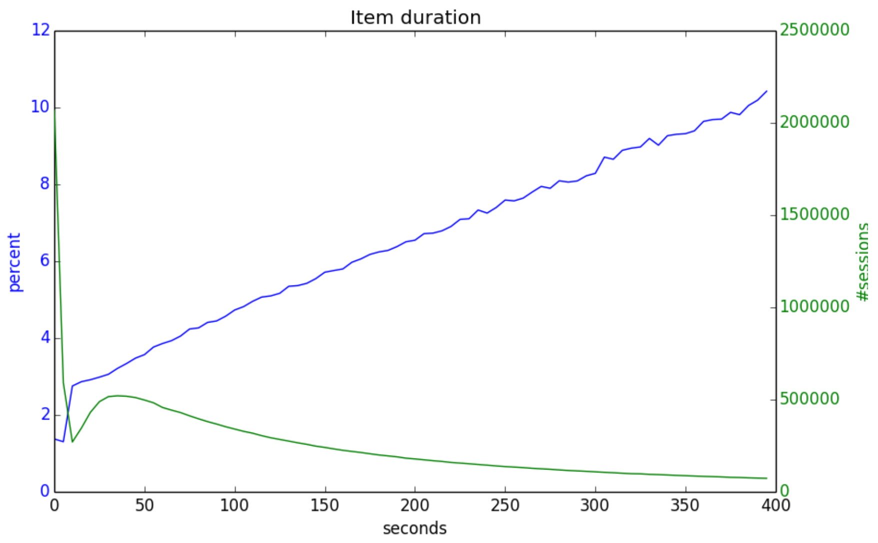


Figure A.1: Relationship between maximum item duration for a session and percent of buy-sessions

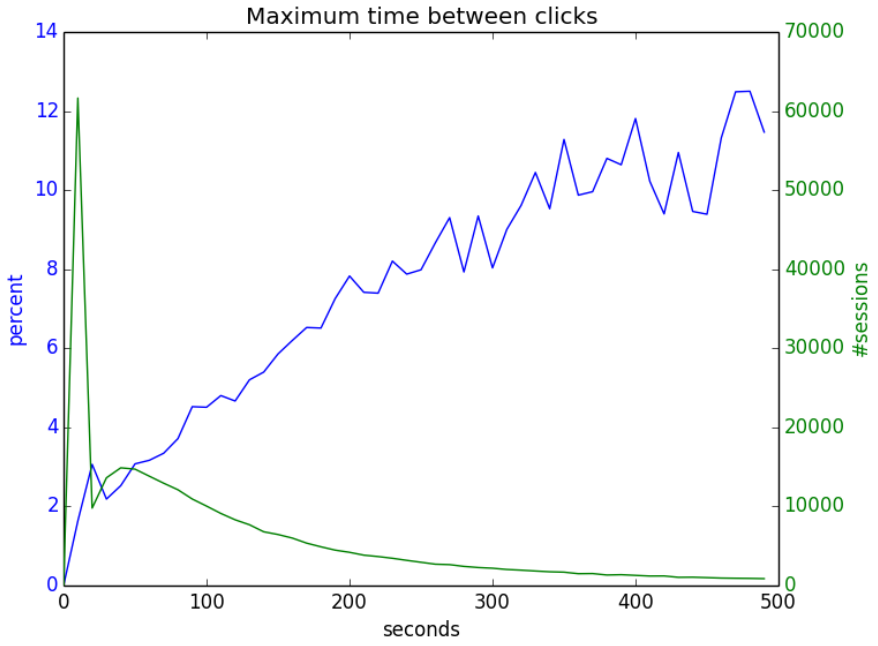


Figure A.2: Relationship between the maximum time between clicks for a session and percent of buy-sessions

We know that the longer a session lasts the higher are the probability of that session ending with a buy. It would be intuitive to think that a session with a longer duration spends more time looking at each item. Could it be that a long time spent watching a single item would increase the probability of a session buying something? In Figure A.2 we have plotted the maximum time spent between two clicks in a session. We observe that the longer the longest time between two clicks in a session are, the higher are probability of that session buying something. The measure of max duration between clicks and session duration have an obvious dependency in that the longer max duration between clicks are, the longer will the session duration for that particular duration be.

A.1.2 Clicks

In Figure A.3 we measure the average clicks per item in the sessions. We can see from this graph that it is more evenly distributed than the two previous graphs, and thus can be harder to get useful information from regarding if a session

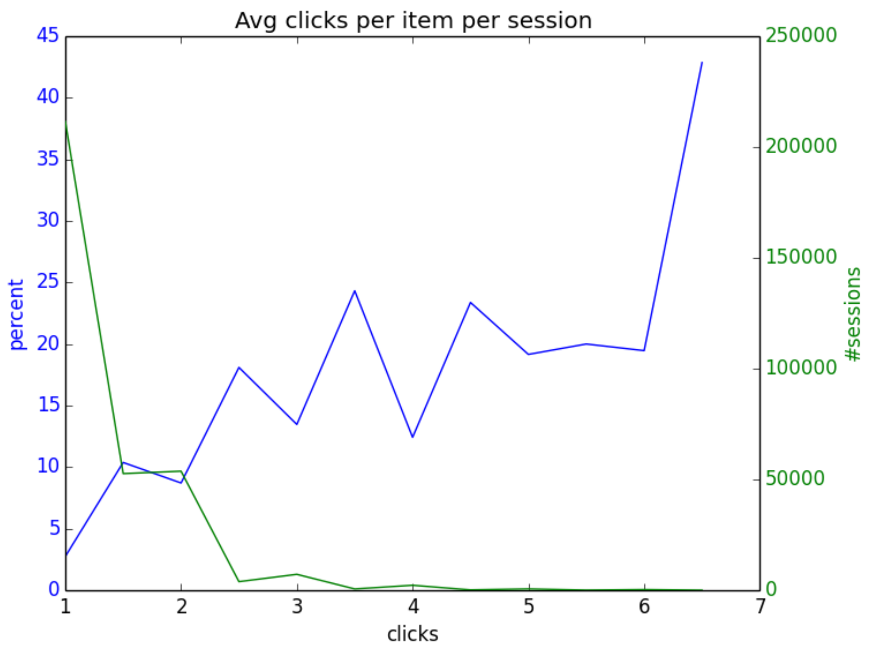


Figure A.3: Relationship between the average number of clicks for a session and percent of buy-sessions

buy something. However we do see a general increase in the probability for the sessions with more than one click one average. This is, of course, strongly related to what we saw in Figure 3.7.

A.1.3 Items

Figure A.4 shows a scatter plot for the relationship between the duration of a session and the maximum probability of buying an item this session has clicked. As each item in the data set has a certain global probability of being bought, the maximum probability used here, is the highest item probability of all the items a session has clicked. We can not see any dependency between how long a sessions lasts, and what the maximum probability of the items they click is. Further we see that most of the sessions have a session duration between 0 to 100 seconds and only click items with probability less than 10 percent.

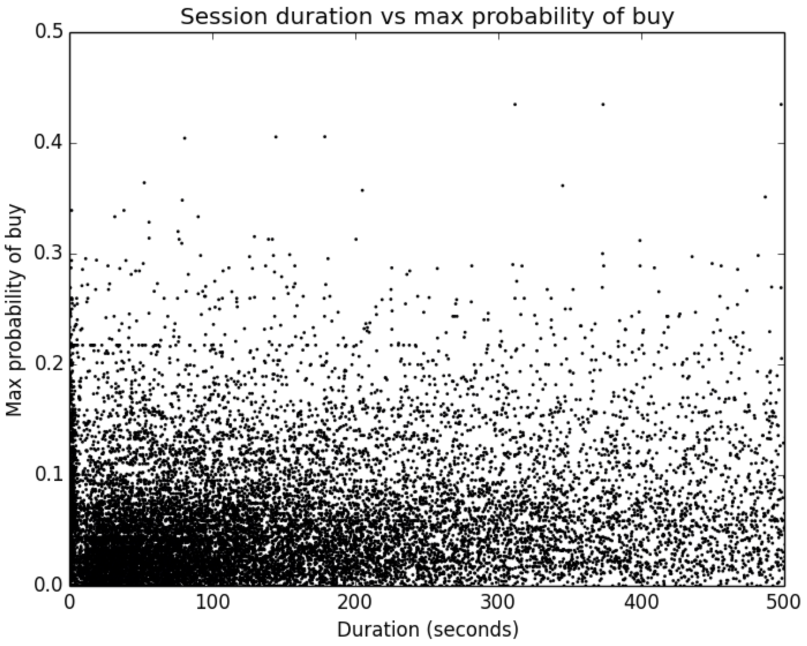


Figure A.4: Frequency of sessions, given maximum probability of an item a session has clicked and the session duration

A.2 Items-bought analysis

Here we present the rest of the analysis conducted for items-bought classification.

A.2.1 Clicks

We observed in Section 3.2 that clicking an item several times without clicking an other item in between was a strong signal for that the session ended up buying something. In Figure A.5 the blue line shows the maximum time a session has spent between two such clicks on the x-axis, and the probability of that session buying the item on the y-axis. If an item was not clicked two or more times consecutively the duration is set to 0 seconds. We observe that such items are bought around 40 percent of the time. Further we see that the probability of buying items of which has been clicked consecutively, increases drastically up to around 50 seconds, and then has a steady increase.

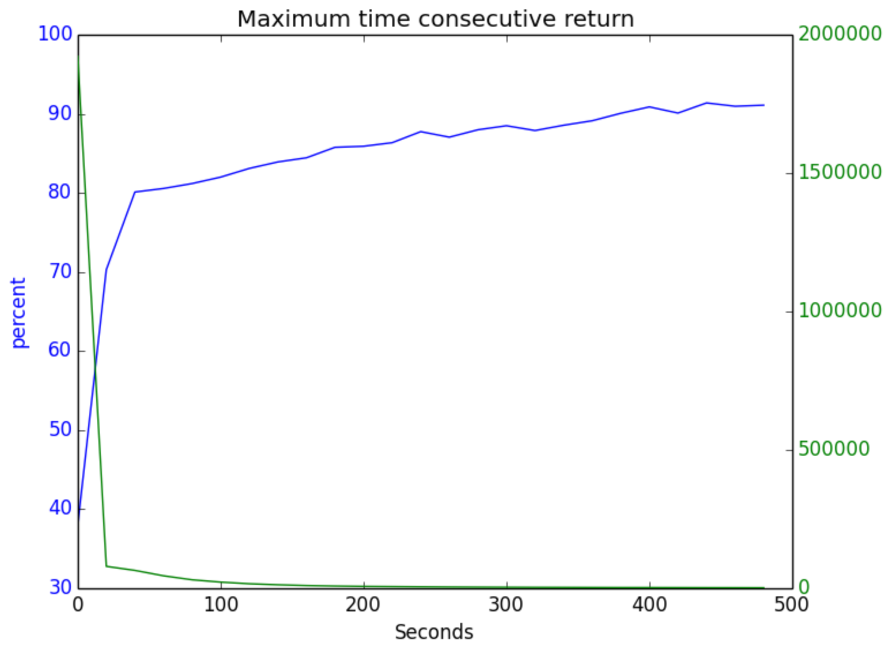


Figure A.5: Percent of items bought, given maximum time between two consecutive clicks

A.2.2 Items

It would be nearby to think that the probability of a session buying an item is strongly correlated with the probability of a session buying an item given that the session bought something. In Figure A.6 we can see that this is the case. However, there are some items that shows us that there can be some differences. There exists items which have been clicked more by no-buy sessions and therefore the percentage of these items being bought increases more when you only include buy-sessions.

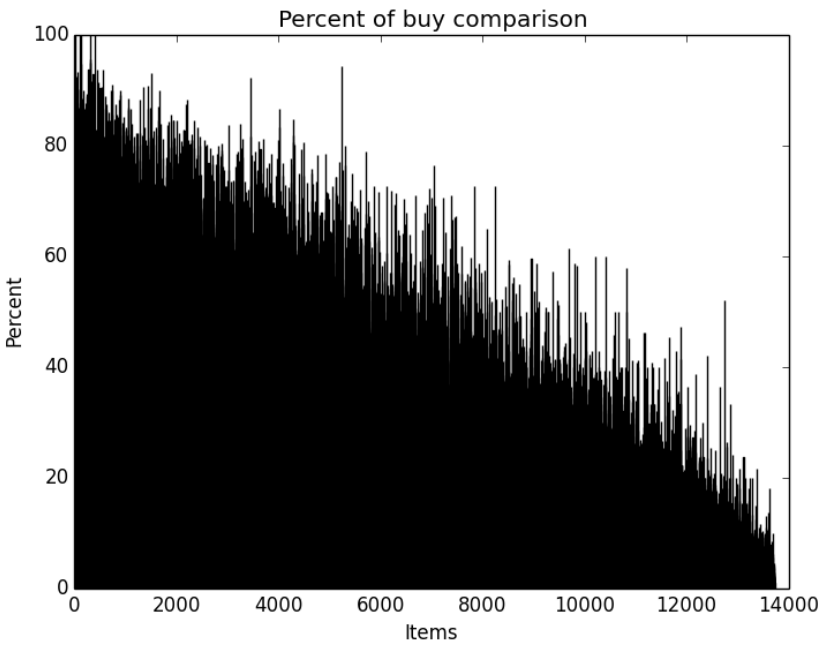
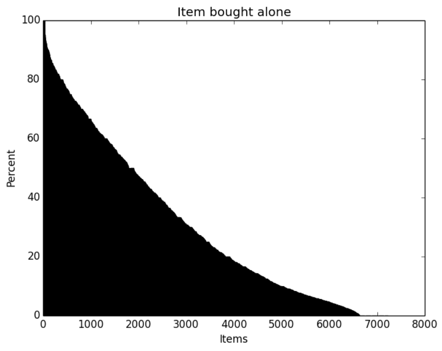
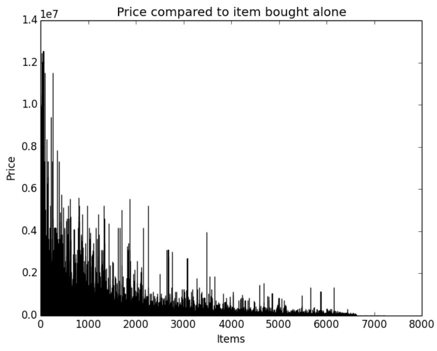


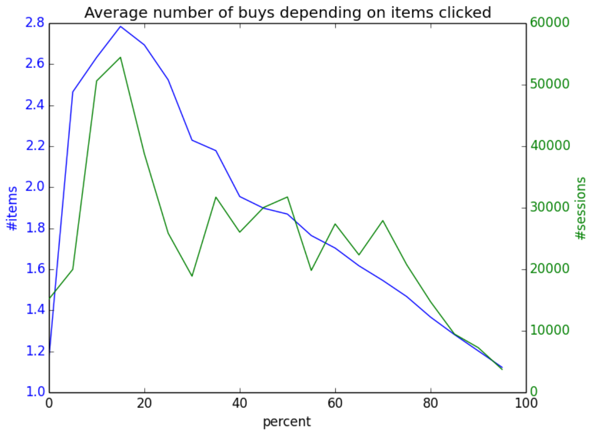
Figure A.6: Percent of buy-sessions buying item given click on item, sorted by percentage of a session buying the item



(a) The percent of buy-sessions that have bought a specific item alone



(b) Item price sorted by percent of buy-sessions that bought item alone



(c) Average number of buys for a session, given the maximum percentage of the session buying an item alone

Figure A.7: Item dependencies

In Section 3.3.1 we saw that the average number of items a session ends up buying depends on when the session occurred. Here we want to explore if certain items also affect the number of buys. In Figure A.7a we have plotted the items on the x-axis and the percentage of sessions that ends up buying that item alone on the y-axis. We observe that there are some items that are bought more often alone than others. A reasonable explanation for this can be the price of the items. One would assume that the more expensive an item is the more probable is it that the session only ends up buying that item. In Figure A.7b we have plotted the items in the same order as in Figure A.7a, but plotted the price of the items on the y-axis instead. We observe that there is a correlation between the price of the item and the probability of a session buying that item alone, supporting our assumption. We conclude this discussion with the results from Figure A.7c. Here we have plotted the maximum percentage of the session buying an item alone(given what items the session has clicked) on the x-axis and the average number of buys on the y-axis. We see that if a session clicks an item that have a high probability of being bought alone, the session ends up buying less items.

Appendix B

Parameters

B.1 Parameters for classification algorithms

Here we present the default parameters used for all algorithms. When other parameters are specified, these are changed. The other possible parameters have the default values stated here.

B.1.1 Random Forest - scikit-learn

RandomForestClassifier(n_estimators=200, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False, class_weight=None)

B.1.2 Random Forest - Weka

Only used for cost-sensitive learning
RandomForestClassifier(debug = False, doNotCheckCapabilities = False, dontCalculateOutOfBagError = False, maxDepth = 0, numExecutiveSlots = 1, numFeatures = 0, numTrees = 100, printTrees = False, seed = 1)

B.1.3 Decision Tree - scikit-learn

DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, class_weight=None)

B.1.4 Decision Tree - Weka

Only used with cost-sensitive learning.
DecisionTreeClassifier(binarySplits = False, collapseTree = True, confidenceFactor = 0.25, debug = False, doNotCheckCapabilities = False, doNotMakeSplitPointActualValue = False, minNumObj = 2, numFolds = 3, reducedErrorPruning = False, saveInstanceData = False, seed = 1, subtreeRaising = True, unpruned = False, useLaplace = False, useMDLcorrection = True)

B.1.5 Logistic Regression - scikit-learn

LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='liblinear', max_iter=100, multi_class='ovr', verbose=0)

B.1.6 Logistic Regression - Weka

Only used for cost-sensitive learning.
LogisticRegression(debug = False, doNotCheckCapabilities = False, maxIts = -1, ridge = 0.00000001, useConjugateGradientDescent = False)

B.1.7 Naive Bayes - scikit-learn

GaussianNB()

B.1.8 Bayesian Network - Weka

BayesNet(BIFFile=", debug=False, doNotCheckCapabilities=False, estimator=SimpleEstimat
-A 0.5, searchAlgorithm=K2 -P 1 -S BAYES, useADTree= False)

B.1.9 Cost-sensitive learning - Weka

CostSensitiveClassifier(classifier='CHOOSECLASSIFIER', costMatrix=2*2 cost matrix, costMatrixSource=Use explicit cost matrix, debug=False, doNotCheckCapabilities=False, minimizeExpectedCost=True, onDemandDirectory=Weka-3-7, seed=1)

B.1.10 Feature Selection wrapper - Weka

WrapperSubsetEval(conservativeForwardSelection=False, debuggingOutput=False, generateRanking=False, numExecutionSlots=1, numToSelect=-1, searchBackwards=False, startSet=", threshold=-1.7976931348623153E308)

Appendix C

Loess Regression

For items-bought classification, we tested Loess Regression with fractions from 0 to 1 with intervals of size 0.1. Each point in the regression for an item was the probability of buying that item that day. A fraction of 0 means that no smoothing is applied, while a fraction of 1 means that all data points are used. In order to see which fraction worked best we measured the score with Logistic Regression classification with all the features we have presented so far. Logistic regression was preferred over Random Forest for this task as it is a more stable classifier, it produces the same result given the same input when classifying. As the items have a big difference in their click distribution, we looked at splitting the items in different data sets and run Loess Regression on each part. In total we ended up with four sub sets: as earlier we have split the items on the number of clicks, one with the probabilities for buying items when clicked once (P_1) and one with the probability of buying if clicked twice or more (P_2); and then these two were split depending on whether the items had more or less than 10 in standard deviation (in terms of number of clicks). Items that in some periods have a lot of clicks, are likely to give relatively accurate probabilities, such as items on sale which we have seen generates a lot of clicks. As there are a period of sales in the last part of the time period, we figured that items with high standard deviation have a higher probability of being on sale in this period, and thus may be better of with a smaller fraction in the Loess Regression.

Splitting the items on standard deviation resulted in four different sets of items we had to optimize the fraction for. Testing all combinations would have been too computationally expensive. Instead we chose to randomly select a fraction for each set and run this 40 times. We found that the best result, yield the highest local score, was reached for P_1 when $f = 0.5$ for items with standard deviation under 10 and $f = 0.1$ for items over. For P_2, $f = 0.7$ for items with standard deviation under 10 was best and $f = 0.1$ for items above 10.

Finding the best fraction to use as parameter for the Loess Regression, consisted of two problems, finding the best for the probability feature used in item-bought classification, and finding the best for the features *max_prob*, *avg_prob* and *least_one_prob* in buy-or-not classification. The fraction of data to use when smoothing the probabilities with Loess Regression depends on the data set you are performing the regression on. A dense data set with a low variance for each day may need a lower fraction than a data set with a higher variance.

Approach	P	R	ROC	local score
Loess_opt	0.725	0.714	0.816	8763

Table C.1: Appendix C: Loess Regression result

Loess Regression does not take into consideration how many clicks that have happened on a given day, only the probability. In this way a day that have had 100 clicks is equally weighted as a day with only 10 clicks. This may be the explanation for its poor results.

Bibliography

[1] RecSys Challenge 2015. Recsys challenge 2015. http://2015.recsyschallenge.com/. Accessed: 2015-06-03.

[2] Avrim L. Blum and Pat Langley. Selection of relevant features and examples in machine learning. *Artif. Intell.*, 97(1-2):245–271, December 1997. ISSN 0004-3702. doi: 10.1016/S0004-3702(97)00063-5. URL http://dx.doi.org/10.1016/S0004-3702(97)00063-5.

[3] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutierrez. Recommender systems survey. *Knowledge-Based Systems*, 46(0):109 – 132, 2013. ISSN 0950-7051. doi: http://dx.doi.org/10.1016/j.knosys.2013.03.012. URL http://www.sciencedirect.com/science/article/pii/S0950705113001044.

[4] Jesus Bobadilla, Antonio Hernando, Fernando Ortega, and Jesus Bernal. A framework for collaborative filtering recommender systems. *Expert Syst. Appl.*, 38(12):14609–14623, November 2011. ISSN 0957-4174. doi: 10.1016/j.eswa.2011.05.021. URL http://dx.doi.org/10.1016/j.eswa.2011.05.021.

[5] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001. ISSN 0885-6125. doi: 10.1023/A:1010933404324. URL http://dx.doi.org/10.1023/A:1010933404324.

[6] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Statistics/Probability Series. Wadsworth Publishing Company, Belmont, California, U.S.A., 1984.

[7] Glenn W. Brier. Verification of Forecasts expressed in terms of probability. *Monthly Weather Review*, 78(1):1–3, January 1950. doi: 10.1175/1520-0493(1950)078\%3C0001:vofeit\%3E2.0.co;2. URL http://dx.doi.org/10.1175/1520-0493(1950)078%3C0001:vofeit%3E2.0.co;2.

[8] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June 2002. ISSN 1076-9757. URL `http://dl.acm.org/citation.cfm?id=1622407.1622416`.

[9] NiteshV. Chawla. Data mining for imbalanced datasets: An overview. In Oded Maimon and Lior Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 853–867. Springer US, 2005. ISBN 978-0-387-24435-8. doi: 10.1007/0-387-25465-X_40. URL `http://dx.doi.org/10.1007/0-387-25465-X_40`.

[10] D. R. Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, 20(2):pp. 215–242, 1958. ISSN 00359246. URL `http://www.jstor.org/stable/2983890`.

[11] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM Comput. Surv.*, 27(3):326–327, September 1995. ISSN 0360-0300. doi: 10.1145/212094.212114. URL `http://doi.acm.org/10.1145/212094.212114`.

[12] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In *Machine Learning: Proceedings of the Twelfth International Conference*, pages 194–202. Morgan Kaufmann, 1995.

[13] Fayyad and Irani. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. pages 1022–1027, 1993. URL `http://www.cs.orst.edu/~{}bulatov/papers/fayyad-discretization.pdf`.

[14] Cesar Ferri, Jose Hernandez-orallo, and Peter A. Flach. A coherent interpretation of auc as a measure of aggregated classification performance. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 657–664, New York, NY, USA, 2011. ACM. URL `http://www.icml-2011.org/papers/385_icmlpaper.pdf`.

[15] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163, 1997. ISSN 0885-6125. doi: 10.1023/A:1007465528199. URL `http://dx.doi.org/10.1023/A%3A1007465528199`.

[16] José Hernández-Orallo, César Ferri, Nicolas Lachiche, and Peter A. Flach. The 1st workshop on ROC analysis in artificial intelligence (ROCAI-2004). *SIGKDD Explorations*, 6(2):159–161, 2004. doi: 10.1145/1046456.1046489. URL `http://doi.acm.org/10.1145/1046456.1046489`.

[17] Jr. Hruschka, EstevamR., EduardoR. Hruschka, and NelsonF.F. Ebecken. Feature selection by bayesian networks. In AhmedY. Tawfik and ScottD. Goodwin, editors, *Advances in Artificial Intelligence*, volume 3060 of *Lecture Notes in Computer Science*, pages 370–379. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-22004-6. doi: 10.1007/978-3-540-24840-8_26. URL `http://dx.doi.org/10.1007/978-3-540-24840-8_26`.

[18] ArjanJ.P. Jeckmans, Michael Beye, Zekeriya Erkin, Pieter Hartel, ReginaldL. Lagendijk, and Qiang Tang. Privacy in recommender systems. In Naeem Ramzan, Roelof van Zwol, Jong-Seok Lee, Kai Cluver, and Xian-Sheng Hua, editors, *Social Media Retrieval*, Computer Communications and Networks, pages 263–281. Springer London, 2013. ISBN 978-1-4471-4554-7. doi: 10.1007/978-1-4471-4555-4_12. URL `http://dx.doi.org/10.1007/978-1-4471-4555-4_12`.

[19] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273 – 324, 1997. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/S0004-3702(97)00043-X. URL `http://www.sciencedirect.com/science/article/pii/S000437029700043X`. Relevance.

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

[21] Miroslav Kubat and Stan Matwin. Addressing the curse of imbalanced training sets: One-sided selection. In *In Proceedings of the Fourteenth International Conference on Machine Learning*, pages 179–186. Morgan Kaufmann, 1997.

[22] Ken Lang. NewsWeeder: learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning*, pages 331–339. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1995. URL `http://citeseer.ist.psu.edu/lang95newsweeder.html`.

[23] Jorma Laurikkala. Improving identification of difficult small classes by balancing class distribution. In *Proceedings of the 8th Conference on AI in Medicine in Europe: Artificial Intelligence Medicine*, AIME '01, pages 63–66, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-42294-3. URL `http://dl.acm.org/citation.cfm?id=648155.757340`.

[24] Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011.

[25] Frank McSherry and Ilya Mironov. Differentially private recommender systems: Building privacy into the net. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 627–636, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-495-9. doi: 10.1145/1557019.1557090. URL `http://doi.acm.org/10.1145/1557019.1557090`.

[26] Jan Hendrik Metzen. Comparison of calibration of classifiers. `http://scikit-learn.org/stable/auto_examples/calibration/plot_compare_calibration.html`. Accessed: 2015-05-13.

[27] Netflix. Netflix. `http://www.netflixprize.com/index`. Accessed: 2015-05-07.

[28] Alexandru Niculescu-mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *In Proc. Int. Conf. on Machine Learning (ICML*, pages 625–632, 2005.

[29] Python. `https://www.python.org/`. Accessed: 2015-03-20.

[30] LauraElena Raileanu and Kilian Stoffel. Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41(1):77–93, 2004. ISSN 1012-2443. doi: 10.1023/B:AMAI.0000018580.96245.c6. URL `http://dx.doi.org/10.1023/B%3AAMAI.0000018580.96245.c6`.

[31] I Rish. An empirival study of the naive bayes classifier. 2001.

[32] S. Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. 1990.

[33] scikit learn. `http://scikit-learn.org/stable/`. Accessed: 2015-06-03.

[34] Hilary L. Seal. Studies in the history of probability and statistics. xv: The historical development of the gauss linear model. *Biometrika*, 54(1/2):pp. 1–24, 1967. ISSN 00063444. URL `http://www.jstor.org/stable/2333849`.

[35] Statista. Statista. `http://www.statista.com/markets/413/topic/457/b2c-e-commerce/`. Accessed: 2015-03-09.

[36] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009. ISSN 1687-7470. doi: 10.1155/2009/421425. URL `http://dx.doi.org/10.1155/2009/421425`.

[37] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *J. Mach. Learn. Res.*, 10:623–656, June 2009. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1577069.1577091`.

[38] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. Introduction to data mining, (first edition). pages 295–297, Boston, MA, USA, 2005. Addison-Wesley Longman Publishing Co., Inc. ISBN 0321321367.

[39] Robin Van Meteren and Maarten Van Someren. Using content-based filtering for recommendation. In *Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop*, pages 47–56, 2000.

[40] Gary Weiss, Kate McCarthy, and Bibi Zabar. Cost-sensitive learning vs. sampling: Which is best for handling unbalanced classes with unequal error costs? In Robert Stahlbock, Sven F. Crone, and Stefan Lessmann, editors, *DMIN*, pages 35–41. CSREA Press, 2007. ISBN 1-60132-031-0. URL `http://dblp.uni-trier.de/db/conf/dmin/dmin2007.html#WeissMZ07`.

[41] Weka. `http://www.cs.waikato.ac.nz/ml/weka/`. Accessed: 2015-02-28.