FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

Company Project # 2015-1428

Martin Nicklas Jørgensen tzk173@alumni.ku.dk

User Behavior Analysis Using Decision Trees

Supervisor: Mikkel Rønne Jakobsen mikkelrj@di.ku.dk

January 17, 2016

Abstract

Simplesite is a danish web-hosting company based in Copenhagen. They recently switched to a freemium based business model and stopped deleting inactive websites entirely in a bid to have more users stay with them. This have lead to a great influx of new customers using the free version of their services, many of which open a free website with their service but do not stay active for very long. Simplesite wishes to improve the situation by first getting users to stay active for longer, and at a later date try to make the active users become paying customers.

In order to gain a better understanding of what make some customers stay and others not, a business problem is formulated and converted to a data science problem; "*Is it possible to find any differences between customers who are active on day 15 and later, and customers who leave before?*" Through the course of the report this problem is worked on using the Cross Industry Standard Process for Data Mining (CRISP-DM); the business problem is converted into a data science problem, I describe the available data and reason about modifications to the datasets before creating and validating different models in order to produce the best possible model. After the model have been created it is evaluated and the process starts over.

The project went through several iterations of the CRISP-DM process, and each iteration is covered and reasoned about. The results are evaluated and suggestions for improvements and implementation ideas are outlined for future work.

The models developed during the project show that for the currently available data, two attributes are the most important for predicting if a customer is retained or not; the number of logins and the number of edits during the first 14 days.

Finally the report contains my reflections on how my education have been used in this company project and how the academic diciplines have I have been taught at DIKU have been applied or expanded during the project.

Contents

1 Introduction

During this project I work with the Danish website/hosting company Simplesite ApS[1] to help analyze user behaviour. This analysis is done through the collection of behavioral data from a base of existing customers and creating decision tree models based on this data. Different types of decision trees and parameters was created and tested to find the configuration that yields the best results and will allow Simplesite to gain the best posbbile understanding about their customers behvaviours.

The report starts with information about Simplesite as a company as well as their product. It then goes on to describe the business problem that I want to solve. After the problem is described, the analysis section shows the Cross Industry Standard Process for Data Mining[10] (CRISP-DM). This process guides the entire process of understanding the business problem to preparing data, creating models and evaluating and deploying the knowledge or models acquired from the project.

In the end of the report I go over the results of the experiments and outline som possible work to improve these results. I also reflect on how some of the skills acquired at DIKU fit into the project as well as what new knowledge I have acquired.

The source-code files from the project can be obtained by emailing me (see frontpage for contact details).

2 Simplesite ApS

Simplesite is a danish website/hosting company that was founded as *Elk Consulting ApS* in 2001 by Morten and Jacob Elk [4]. In 2003, the company changed its name to *123Hjemmeside ApS* in order to reflect the product they delivered. Over the years, as 123hjemmeside expanded into new countries, the name was translated into the local equivelant for each version of the site. In 2014, 123hjemmeside adopted its secondary name *Simplesite ApS* as its new primary name in order to unify all it its brands under a single name.

The information in this section is gathered from [4], interviews with and from having worked in the operations department for 5 years.

2.1 Product

Simplesite produce and maintain an inhouse website Content Management System (CMS) and operate a hosting location where this CMS runs. Apart from the full paid version, customers can register for a free account allowing them to host a website that is edited through the CMS. The free accounts are under restrictions on number of pages, images and videos that can be added to the site. Customers can then change to a paid subscription which allows them to have more pages, images and videos on their website.

Simplesite also offer additional services to their customers that can be bought for an extra fee if you are already a paying customer, this includes the ability to have a domain attached to your website, a webshop as well as additional pages, images and videos.

The product is hosted partially on Simplesites own hardware in an offsite location, and using a number of cloud services to provide faster response times for certain data types.

2.2 Departments

The employees work from three different offices located in Denmark, Bulgaria and Serbia, but also have people working from other locations around the world.

- **Administration** - Situated at the ground floor of the Copenhagen office, this department contains the HR functions as well as finance.

[1] `http://www.simplesite.com`

- **Sales** - Also on the ground floor of the Copenhagen office, sales consists of full time employeess and student helpers. The primary task is to sell the product, currently through localized ad management.
- **Product & Communication** - The department sits on the upper floor of the Copenhagen office and is responsible for planning new features in cooperation with the developers as well as manage content on Simplesites own websites. P & C also manages communication such as newsletters and localized ad texts.
- **In-House Development** - Sitting next to P & C on the upper floor the developers are responsible for implementing new features as well as maintenence, analytics and bug fixing of the product.
- **Operations** - Daily operations are handled primarily by the company CTO, Thomas, as well as 2 part time students. Operations sits on the upper floor in Copenhagen next to the developers.
- **Support** - Since the product is offered in several languages, a supporter is hired per language in a part time basis. All supporters work from home but get together once a month for a status meeting and to make sure new knowledge is shared and that relevant information can be given from the regular departments.
- **Remote Dev 1** - A small number of developers work in Bulgaria and have their own office in the Sofia, they are offered machines in the Copenhagen office they can VPN into and use for work. This allows the operations department to work from Copenhagen and still service the remote developers.
- **Remote Dev 2** - A number of developers work from Serbia, like the remote developers in Bulgaria, they also VPN into the Copenhagen office and work on machines maintained by the regular operations employees.
- **Miscellaneous** - Simplesite also occasionally employs external specialists or consultants. Depending on the need, they will either work in the Copenhagen office or from some remote location using VPN.

2.3 Company Structure

Figure 1 shows the overall organization of Simplesite. For simplicity I have merged the remote development offices into the "regular" development department since they occupy the same space in the organization, the only difference is the geographical seperation.

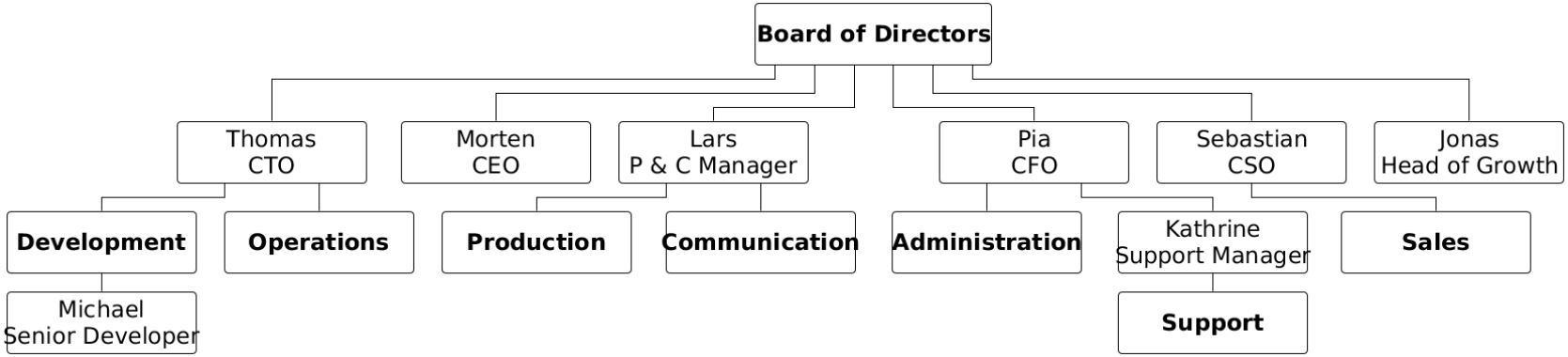

Figure 1: Organizational diagram for Simplesite ApS, text in bold represents departments. Last names are omitted to keep the figure small.

3 Problem Description

Before 2014 123hjemmeside offered paid subscription service that allowed customers to create and host a website using the provided CMS. Customers could get a time-limited trial but would have to buy a subscription to continue using the service afterwards. If a customer stopped paying their subscription or did not wish to continue after their trial, their website would be kept offline for a short while, then deleted.

When 123hjemmeside changed its name to Simplesite in 2014 it also changed its business model. They went from being purely paid subscription service to allowing everyone to create as many websites as they wished for free, with no time limitation. People using the free subscriptions have access to most functions of the CMS, excluding functions like personal domains and webshops as well as putting a limit on the number of images a website can contain. In order to use these features, a customer will have to sign up for a paid subscription. This model of free and premium products is known as freemium ([1, p. 6] and [8, p. 1]).

After the change of model, Simplesite could see a huge increase in the number of free websites being created, but not a corresponding increase in paid subscriptions. They came up with the idea of recording how customers use the site and gather the data to build decision trees ([3] and [9]). The decision trees should not only be used to classify customers based on behaviour, but also to let Simplesite learn what events in the users' life cycle that increase the chance of them becoming paying customers.

Initially Simplesite is interested in exploring if there is any differences between the users that still log in 15 or more days after they have created their account (Simplesite calls these customers *retained*) and customers that simply make a site and leave or forget about it. This mapping of important "actions" that some customers take is supposed to be used later in a mail procedure that will automatically send out emails to new customers prompting them to perform these actions that are known to cause customers to be retained.

The longterm goal is to be able to send out personalized emails to customers in an attempt to keep the active on the site. These emails could contain tips and tricls for using a feature that the user have not yet touched ot to attempt to prompt them into writing a new page or blog entry.

3.1 Requirements

Simplesite have already created some code in R[2] so any continued work should be done in the R language. I have also been added to a Github repository containing analytics code and a preferred structure and coding style that should be adhered to.

Two datasets are also available containing different information gathered from the live system. These two datasets should form the basis for all the data analysis performed. An additional dataset is being constructed during the project, but due to the time it takes to populate it with enough observations to be meaningful, it was not be finished before the project ended.

3.2 Success Criteria

The project have 4 success criteria (that should all be fulfilled to some degree):

1. A decision tree model for classifying customers should be created.
2. New knowledge about customer lifecycles acquired from the model, in particular, do *retained* customers have something in common.
3. A prototype R script that can automatically build/create the model instance from new customer data should be created.
4. A method for using the model should be designed or reasoned about.

[2] `https://www.r-project.org/`

4 Problem Analysis

In order to work with the the problem described in Section 3 I used a workflow called called the *Cross Industry Standard Process for Data Mining*, shortened to CRISP-DM (as described in [7, p. 26], originally from [10]). An overview of the process can be seen in Figure 2. The following subsections will go over each phase in the process, explaining what the purpose of the phase is and cover my work during that phase of the project.

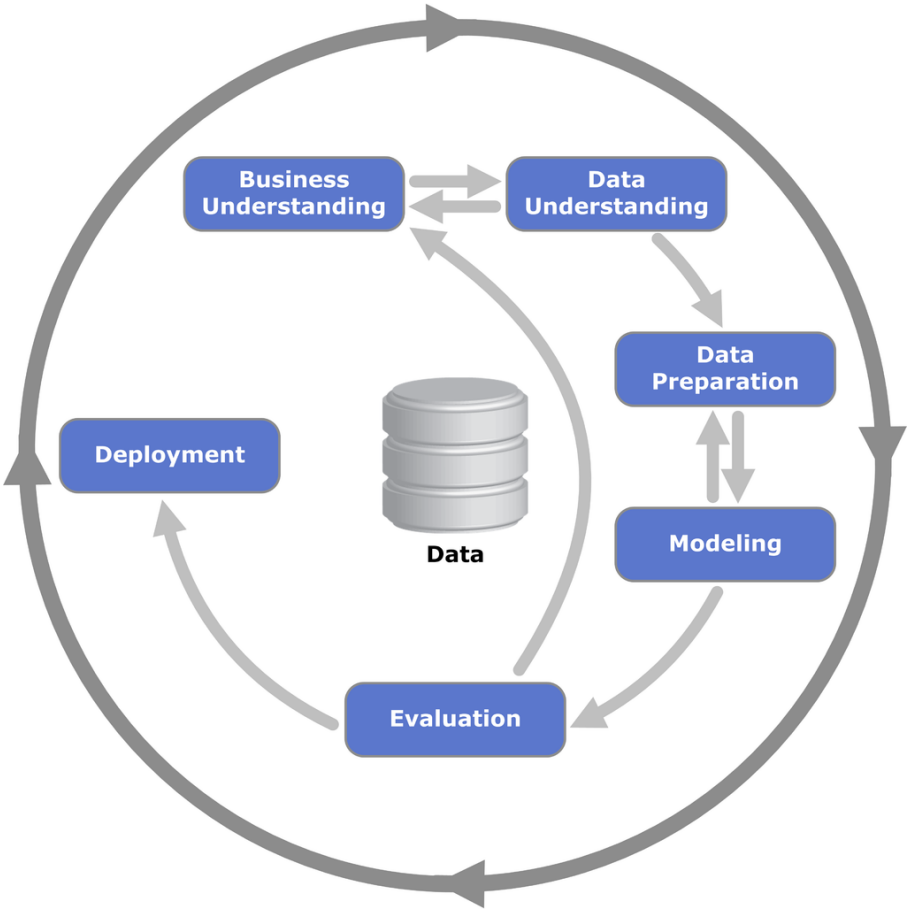

Figure 2: Diagram of the CRISP-DM method. Source: [6].

Simplesite have performed most of the *business understanding* step so the work I performed is placed in *data understanding*, *data preparation*, *modelling* and *evaluation*. Due to the short duration of the project, there have not been any time spent with the *deployment* phase.

4.1 Business Understanding

The business understanding phase is the entry point for most data science projects, it covers the process of taking a business project, and converting it to a data science project. My work in the phase have been very limited, since Simplesite had a very clear idea of what they would like to investigate first. The initial business problem for this project can be described as "*We have a large number of free users, but a lot of them do not stay with the product. Can we find out why?*". From this business problem it makes sense to ask "*Can we find a difference between the customers that stay and those who don't?*", more specifically is there a difference in the way the customer behave when using the site? And that question is what this project is being performed from.

4.2 Data Understanding

The data understanding phase of data science is the second phase. During this phase you look into what data is available and collect whatever information might be relevant for the data science problem. During this phase you will also identify data quality problems and gain familiarity with the data in order to help form a better hypothesis. It is worth noting that weaknesses in the dataset are not removed until the next phase, data preparation, so in order to make the report more readable, any arguments for removal or change will be listed in Section 4.3 instead of here.

The basis for the analysis is 2 datasets created by Simplesite: `EngagementData` & `CustomerJourney`. Table 1 and Table 2 names the features of each dataset and what they represent. Both datasets contain users created between September 1st 2015 and September 30 2015. The data is recorded for all users after this point as well, but the larger dataset also increases memory consumption, and makes it unwieldy. The training datasets both contain 463716 observations, each corresponding to a user created in this timespan.

Another copy of each dataset was created from the database pulling data from users created between October 1st 2015 and October 30 2015. These datasets contains 495390 observations and will be used as test data.

Attribute Name	**Attribute Data**
islogins1	Bool, true if: one or more logins for the user.
islogins2	Bool, true if: two or more logins for the user.
islogins3	Bool, true if: three or more logins for the user.
islogins4	Bool, true if: four or more logins for the user.
isedit30m	Bool, true if: User edited site within 30 minutes of creation.
isedit24h	Bool, true if: User edited site within 24 hours of creation (excluding the first 30 minutes).
isaddpage30m	Bool, true if: User added a new page within 30 minutes of creation.
isaddpage24h	Bool, true if: User added a new page within 24 hours of creation (excluding the first 30 minutes).
isimgupload30m	Bool, true if: User uploaded their own image within 30 minutes of creation.
isimgupload24h	Bool, true if: User uploaded their own image within 24 hours of creation (excluding the first 30 minutes).
iseditdesign30m	Bool, true if: User edited site within 30 minutes of creation.
iseditdesign24h	Bool, true if: User edited site within 24 hours of creation (excluding the first 30 minutes).
customerid	Integer value with the customers unique ID.
marketname	String with the market the user came from (US, TR, DK etc.)
siteverkey	String with what version of the site the user is created in (US, TR, DK etc.)
ispayer	Bool, true if: The customer have a paid subscription.
culturekey	String with language information for the site (en-US, fr-FR etc.)

Table 1: Features found in the `EngagementData` dataset.

Attribute Name	Attribute Data
customerid	Integer value with the customers unique ID.
logins14	Integer, number of times the customer logged in the first 14 days (week 1-2 after creation).
logisnw2w4	Integer, number of times the customer logged in in week 3-4 after creation.
edits14	Integer, number of times the customer edited a page within the first 14 days.
iscjtrial	Bool, true if: Always true, everyone starts as a trial.
iscjonboarded	Bool, true if: edits14 ≥ 1.
iscjactivated	Bool, true if: edits14 ≥ 3.
iscjengaged	Bool, true if: edits14 ≥ 6 and logins14 ≥ 2.
iscjinvested	Bool, true if: edits14 ≥ 15 and logins14 ≥ 6.
iscjretained	Bool, true if: logisnw2w4 ≥ 1.
isimgupload1d	Bool, true if: Customer uploaded an image within the first 24 hours of being created.
iseditdesign1d	Bool, true if: Customer edited the design within the first 24 hours of being created.
isaddpage1d	Bool, true if: Customer added a new page within the first 24 hours of being created.
isedit1d	Bool, true if: Customer edited a page within the first 24 hours of being created.

Table 2: Features found in the `CustomerJourney` dataset.

4.3 Data Preparation

The data preparation phase is where all the data weaknesses we found in the previous section is removed and all the raw data is collected into the final dataset. At the end of this phase the dataset should contain all the data the models will need and be ready to be "fed" to the model for training and validation.

The initial goal is to find customers who are retained (*iscjretained* = `True`), and to see if there is some pattern that Simplesite can use to try to guide other customers down, in order to increase the number of retained customers. With this in mind there is some attributes of the datasets that will not be helpful, either because they cannot be controlled/changed, or because they do not make sense. The following is a list of attributes removed from the `EngeagementData` dataset during work, along with the reason for the removal.

- *islogins1* : Removed because the knowledge about number of logins is contained in the *logins14* feature from the *CustomerJourney* dataset.
- *islogins2* : Same as *islogins1*.
- *islogins3* : Same as *islogins1*.
- *islogins4* : Same as *islogins1*.
- *marketname* : Removed since we are unable to get a customer from a different market, we are insterested in variable we can change for each customer.
- *siteverkey* : Same as *marketname*.
- *ispayer* : Removed because it is an alternative target variable, it does not say anything about how the user behaves, other than they are indeed a good customer.
- *culturekey* : Same as *marketname*.

The following is a list of attributes removed from the `CustomerJourney` dataset during work, along with the reason for the removal.

- *logisnw2w4* : Removed since this attribute is in the definition of our target variable *iscjretained*.
- *iscjtrial* : Removed since it is always true.

- *iscjonboarded* : Removed since it serves as an alternative target variable. It is set by Simplesite and does not say anything about the user behaviour that is not already present in other attributes.
- *iscjengaged* : Same as *iscjonboarded*.
- *iscjinvested* : Same as *iscjonboarded*.

In both datasets the *customerid* attribute is kept and used to join them into one training dataset, after this join operation the *customerid* attribute is removed.

The files *src/dataset_ training.tsv* and *src/dataset_ test.tsv* contains the data from these final versions of the datasets.

4.4 Modelling

During the modelling phase the models are applied to the dataset constructed in the data preparation phase, and parameters are adjusted to give the best results possible. Some models might require that the data is in a specific form, so it might be necessary to step back to the data preparation phase.

4.4.1 Tree Type

While researching R, two distinct type of decision trees came up; an implementation of regular decision trees that closely follow the method described in [3], as well as conditional inference trees[5], which I will shorten to *ctree* in this report.

Both models produce binary trees that can be used to solve classification problems. Each node in the tree represents a variable and each edge out of the node contains a "case" that tells something about the variable (is it true/false, ≥ 5, and so forth). Each leaf of the tree is representing a class, and there can be several leafs with the same class, they just represent different characteristics of the same class.

The main difference between a regular decision tree and a ctree is how it is created. A regular decision tree will choose to split using information measures such as seen in Quinlan et. al.[9, p. 89], whereas the ctree framework will split based on the relationship between the target feature and the covariates based on conditional distribution. Given a fresh dataset, a regular decision tree will make the first split on the attribute where the resulting split gives the two largest possible subsets on each edge. A conditional inferance tree on the other hand will split on the covariate/attribute with the strongest relation to the target variable.

4.4.2 Model Type Precision

In order to select which tree type to use I performed an experiment that used 5-fold cross validation[2, p. 32] to test the accuracy of the models for different depths. The dataset used for this experiment was the combination of the *EngagementData* and *CustomerJourney* datasets, as described in Section 4.3 (excluding the customerid column.) The code for this test can be seen in Appendix B.1 Figure 1.

Max Depth	`rpart` **Accuracy**	`ctree` **Accuracy**
4	94.2799 %	94.27990 %
8	94.2799 %	94.31958 %
12	94.2799 %	94.36638 %

Table 3: The mean accuracy for the different 5-fold cross validation runs.

The results shown in Table 3 indicates that for our particular dataset, the accuracy difference for the two models is very insignificant. They both predict correctly in around $94,3\%$ of the cases with a difference of less that $0,1\%$ between the best and worst performer.

An interesting observation from the data above is that the `rpart` model have the same accuracy for all three runs indicating that the model created does not change even when it is allowed to grow more complex. After plotting the models from each step, I discovered that it did not grow beyond a depth of 2 and based itself solely on the `logins14` variable.

4.4.3 Choosing a Model

Based on the two criteria highlighted in in Section 4.4.2, I selected to go with the `ctree` package, due to the better accuracy in the experiment and the fact that the model seem to present more options when you allow it to grow more complex. This is a desirable feature since the project is not just about creating a model in the computer, but also to let Simplesite learn about their customers behaviour.

4.4.4 Formula and Tree Depth

In section 4.3 I discarded a number of features in the dataset and reasoned about why they would not be usable for our purpose. This leaves 14 features as well as the target variable, `iscjretained`. Initial work with the trees showed that the algorithm used for construction of the ctrees based a lot of splits on the *edits14* and *logins14* features. This makes sense because they are the only numeric variables in the dataset, the rest of the variables are booleans that indicate if some event transpired. This allows the algorithm to make splits based on whether or not the variable is within some range (`logins14` ≥ 5 for example.), these splits can be performed repeatedly to partition the set into more granular partitions based on a single variable. On the other hand with a boolean attribute you cannot partition more than once (is it true/false) since repartioning the true group based on weither the attribute is true or false again does not make a difference.

In order to test the impact that these two features have on the accuracy on the model I performed a number of 5-fold cross validation runs using different formulas (See the formulas in Table 4) for the tree creation as well as different maximum depths for the ctrees. The code for the experiment can be seen in Appendix B.2 Figure 2. The results of the experiment can be seen in Table 4. The formulas are read as `target_varible ~ features_to_predict_from`, the topmost formula with a dot instead of the feature list simply means "all features".

Formula	**Max Depth**	**Mean Accuracy**
`iscjretained ~ .`	4	94.27990 %
	6	94.29672 %
	8	94.31958 %
`iscjretained ~ edits14`	4	93.50055 %
	6	93.50227 %
	8	93.50119 %
`iscjretained ~ logins14`	4	94.27990 %
	6	94.27990 %
	8	94.27990 %
`iscjretained ~ edits14 + logins14`	4	94.27990 %
	6	94.28465 %
	8	94.29414 %

Table 4: Mean accuracy of different formulas and tree depths using 5-fold cross validation.

The results show that while it makes a small positive difference percentage wise to include all the features, it is less than 1% which as a percentage is not too much, but the model is supposed to be applied to all new users once it is fully implemented, which is more than 400.000 users monthly (Simplesite confirmed that the september dataset I have been using is representative of the number of new pages), where 1% is still 400 potential customers that could've been helped to

stay active but might now be lost, it might also be 400 customers that indended to pay, but due to extra email might now consider to find a provider that sends less unsolicited[3] emails.

Furthermore using all the features also lets Simplesite learn more about what actions (apart form logins and edits) that the models show to be related to retaining customers.

From the results in Table 4 also shows that increasing the maximum tree depth with 2 raises the accuracy by approximately 0.02%. plotting out the trees in question shows that it is because the *logins14* and *edits14* features dominate the first many levels of the tree, so the remaining boolean features only show on the lower levels.

4.5 Evaluation

In the evaluation phase of the CRISP-DM process, the model(s) produced in the earlier step is carefully examined and evaluated; i.e. does it solve the business problem? By the end of this phase a decision should be made wether the data mining results can be used, or if more iteration is needed. Even if the model is deployed, it is still possible to continue the process of iteration to create a better model or discover more things about the data or business problem.

4.5.1 Dataset Bias

While running the tests that produced the data in Table 4 I noticed that while the accuracy stayed around $93 - 94$ % for all the runs, the model did not have the same precision when predicting `TRUE` as it does when predicting `FALSE`. The model would be wrong when predicting a customer to be *retained* one time out of three, giving it an accuracy for predicting `TRUE` of around 66.666 %. When guessing `FALSE` on the other hand; the model would have an accuracy of around 95 % of the time. The data for these calculations can be seen in Table 8, Table 9 and Table 10 in Appendix A.

Looking at the datasets it looks like this trend is caused by the number of observations for each class. Both the training and test dataset contains more than ten times as many unretained customers as retained, the exact numbers can be seen in Table 5. This essentially returned me to the data understanding phase of the CRISP-DM process, but for the sake of continuity in this reportm the data preparation, modelling and evaluation of this extra iteration will be written in this section.

Dataset	`TRUE`	`FALSE`
Training	30358	433358
Test	40731	454659
Equal	30358	30358

Table 5: The distribution of the *iscjretained* target variable classes in the different datasets.

In order to try and combat the bias in the dataset I constructed a new dataset I will refer to as dataset *equal*. This new dataset contains all of the observations from the training dataset who is *retained*, and a random sample, of the same size, of the observations who is not.

Like in Section 4.4.4 I created a new set of tests (the source code for these tests is available in Figure 3 in Appendix B.3) which would test the new accuracy of the model with varying tree depths. Using this new test, the overall accuracy for drops to around 82 % (81.6753 %, 81.9569 % and 81.9438 %) for maximum depths of 4, 6 and 8, for the detailed numbers, please refer to Table 11, Table 12 and Table 13 in Appendix A.

The new dataset and model does better at guessing retained customers with a mean accuracy of around 82 % for both `TRUE` and `FALSE`. While this increase in accuracy for retained customers is roughly equal to the decrease in accuracy for unretained customers when looking at the percentages, the data that the model should predict on contains more unretained customers than retained. So the drop from 95 % to 82 % in unretained accuracy represents a lot more users that

[3]The email is not strictly requested, but the customer have agreed to recieve emails from time to time.

the increase in retained accuracy from 66.666 % to 82 % when referencing the distribution in Table 5. This means that this model will misclassify a lot more unretained users as retained compared to the improved ability to correctly classify retained customers as retained.

4.6 Deployment

In the deployment phase of the process, the results from the data mining is used to either implement changes to the system that the modelling uncovered or implement the model itself into a workflow that allow it to be resued with new data later.

Due to the time restrictions and the fact that the results of the project is published for the first time in this report, the deployment phase of the CRISP- DM process will be taking place after the end of the project. For the final results and ideas for future work, see Section 5.

5 Results

This section will contain the results of the project, both in term of the final models created for classifying whether or not a customer will be retained, but also recommendations for future work that could improve the solution.

5.1 Modelling Results

In Section 4.5 we saw that when removing the bias towards unretained observations from the dataset, the overall accuracy of the model suffers severly, with that in mind, I chose to train the future models on the full dataset, since this dataset also contains what the user have already "done", even if a customer is mistakenly identifyed as a user that will not be retained, Simplesite can check via this data that they do not accidentally send out emails with tips and tricks that the customer already know.

By using the full training dataset and formula with the highest precision from Table 4 I have run the tests by training a model on the entire dataset (witout cross validation) and tested the resulting model on the full test set. In order to make sure that the model did not overfit, I tested with three different maximum depths to see if there would be any significant difference due to overfitting, the code for running this experiment can be found in Appendix B.4 Figure 4.

Maximum Depth	**Accuracy**
4	92.84039 %
6	92.84846 %
8	92.75823 %

Table 6: The results of the final datarun when training on the full training set and trying to predict the entire test set.

Table 6 shows that the difference between allowing the tree to grow 6 levels deep, compared to 4 in terms of accuracy is less than one hunredth of a percent. The results also show that allowing the tree to grow to 8 levels deep causes a slight loss of precision that is most likely due to an overfitting for the training data.

Due to the size of the trees I was only able to include the tree with a maximum depth of 4 in the report, since the larger ones does not fit on A4 paper or on my screen[4] but they can be reproduced by running the code from Figure 4. Figure 3 shows the resulting model when drawn on the screen. Each node in the drawing corresponds to an attribute, or a choice, and each edge out of the nodes corresponds to a value or statement about the attribute. The leafs contain 2 pieces of information; the n variable which is the number of observations in the training set that belongs to this leaf, as well as the propability of the observations in that leaf to be false (unretained). If

[4]Displaying them on a 2560×1440p screen results in nodes and leaves overlapping.

we use the leftmost leaf as an example, there is 275873 observations that belong in this leaf, and the propability that any of the observations that land there is unretained is 1, with a 0 propability that they are retained.

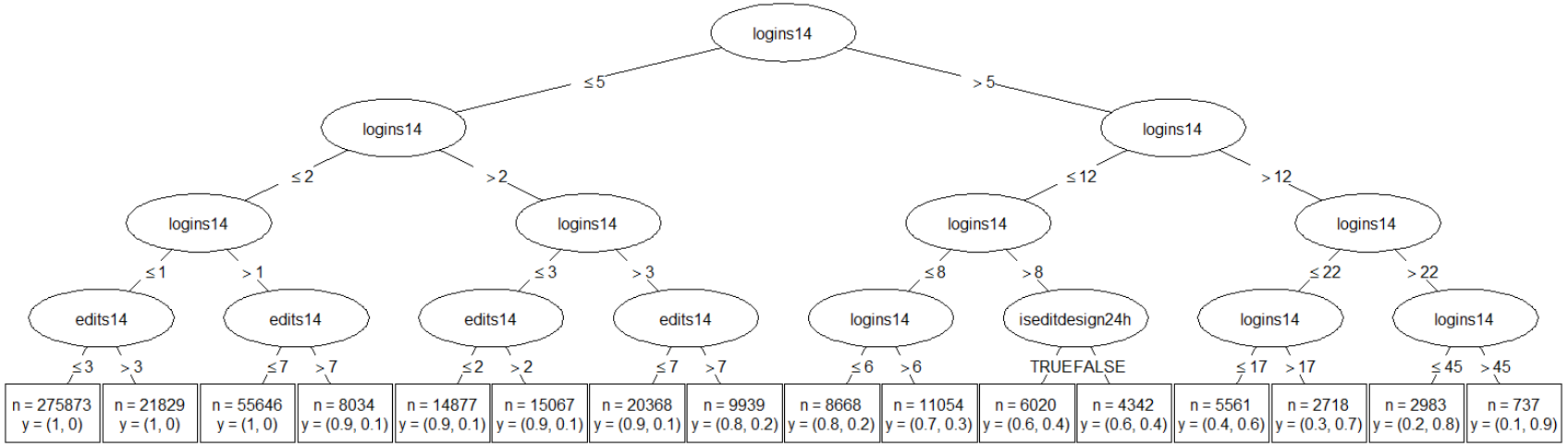

Figure 3: The comditional inference tree produced by the code when using a maximum depth of 4.

Figure 3 also shows that the *logins14* variable is taking up a lot of the nodes. This could indicate (based on our data) that one of the most important factors in retaining a user, is to make them form the habbit of logging in. Users that log in more than 12 times during the first 14 days have a propability of at least 0.6 to be retained.

In order to learn if any other variables can impact the customer retention I performed a test with the *logins14* attribute removed from both the training and test datasets. (This moves us back into the data preparation phase of the CRISP-DM process). The accuracy data from the test can be seen in Table 7.

Maximum Depth	**Accuracy**
4	91.79051 %
6	91.74852 %
8	91.74388 %

Table 7: The results of the final data run when training on the full training set excluding the *logins14 variable* and trying to predict the entire test set.

The result of excluding *logins14* is a drop in accuracy of around 1 %, but when looking at the tree produced we might gain more knowledge. The tree produced by limiting the depth to 4 can be seen in Figure 4.

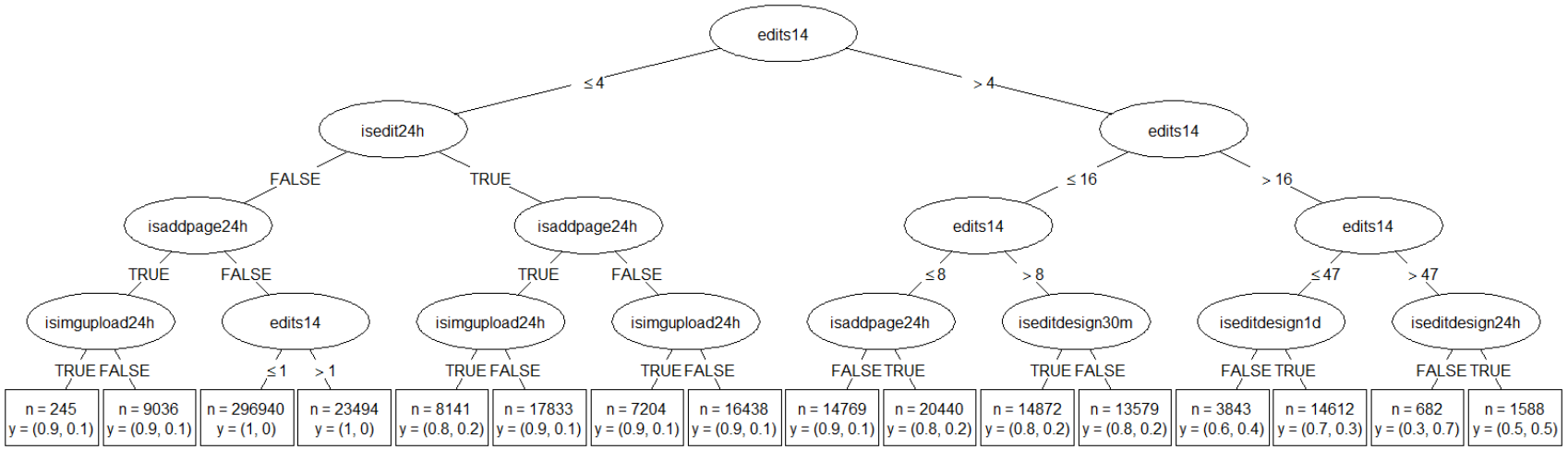

Figure 4: The conditional inference tree produced by the code when using a maximum depth of 4 and exluding the *logins14* attribute.

Looking at Figure 4 we can see a lot more of the attributes have been selected, but also that the propability of retention seem to be rather low in most of the leafs. The 4 leafs with the best retention span from 0.3 to 0.7 and all come from users that have more than 16 edits within the first 14 days of their time with Simplesite.

Based on the two resulting trees in Figure 3 and Figure 4, it is hard to pinpoint a series of specific actions that makes retained/retainable users easy to see. The two most significant traits (as we could see by the numbers in Table 4) seems to be the number of logins as well as edits during the first 14 days of the users time with Simplesite.

5.2 Continued Work

5.2.1 Dataset

In order to further improve the models a new dataset could be compiled, based on this project it looks like "counter" attributes give a lot more information than boolean attributes; a dataset that also counts the number of image uploads or edits within a more fine grained time window might shed more light on the behvaiour differences of retained and unretained customers.

5.2.2 Automation

This project includes code that allows an R program to load data from a clear- text file and use it to automatically create a model from the dataset. Simplesite already have code that allows the programs to pull the data directly from the database, so as long as the script is run regularly it can pull fresh data and create an up-to-date model.

A similar approach could be used for uploading the predictions to a database after the model is created and the new user data have been acquired. Since the model is part of a new program to send out more personalized emails to customers, it can be used to find customers that have very low login or edit counts, or are somehow deviating from the patterns we see in retained customers. When the model classifies a customer that can be prompted by an email, it can use the same SQL driver it used to get user data to add the customer to the email queue.

The following procedure illustrates this idea:

1. Get a training set of customers that are at least a month old.
2. Get a set of customers to predict on, these could be created 14 days ago and to the date of this execution.
3. Create a conditional inference tree model based on the training set.
4. Use the model to predict which of the new customers are "on the right track".
5. Discard these customers, we don't wish to interfere with them.
6. For the remaining customers in the list: based on knowledge from the database, add them to the email queue and send them an appropriate email, for example: "Did you know you can create a free image gallery on your Simplesite?" or similar.

The implementation and regular use of this procedure would hopefully lead to increased retention of customers with free accounts, and with some adoption and a change of the target variable also be able to predict on whether or not a customer will become apaying customer. This model could be applied at a later stage so new users are fed to the current model that predicts retention, and customers that are retained can be fed into a model that predict on becoming a paying customer, further expanding the idea of guiding potential customers along a behavior that is known to have a high propability of becoming a paying customer.

5.3 Competencies and Methods

In this section I will briefly reflect on how the skills and knowledge I have acquired during my studies was used during the project as well as any skills I have acquired during the project with Simplesite.

The major part of this project was rooted in the fields of Data Science and Machine Learning. I had no experience or knowledge of Data Science but much of the knowledge I have acquired through my studies was applicable since data science uses a lot of techniques from other fields in computer science. The CRISP-DM process from Figure 2 stems from data mining and is strongly reminiscent of iterative software development, and data science directly apply machine learning techniques such as model validation. I also learned new things from studying data science, such as "feature engineering", which is the dicipline of creating new features for a dataset using only the existing features.

I also expanded my knowledge of machine learning with decision trees. It is a model that was not covered during the "Statistical Methods in Machine Learning" course at DIKU, but it is a very useful model to use if the model should also be readable by humans.

Another skill I learned from DIKU that came in handy was the ability to quickly pick up on a new programming language and learn to use it, since the project required me to learn the R programming language and start using it immediately due to the time constraints of the project. This particular ability is not explicitly taught at DIKU, but most students pick it up as many courses introduce a new programming language specifically for the course, with "Advanced Programming" using three different languages over a 2-3 month period.

Two other skills that are not specifically tought at DIKU but most students pick up are how to work with versioning and collaborative systems such as SVN or Git, as well as the ability to quickly gather domain specific knowledge through articles and textbooks. Both came into use during the project with Simplesite, since the code is hosted on Github, and I had no existing knowledge on Data Science and had to learn it from scratch.

6 Conclusion

After a product change Simplesite was facing a rapid influx of users for their free services whom did not stay active on the site after 14 days of use. In order to gather some knowledge, Simplesite whished to build a decision tree based model that could be used to predict if a user would be retained after the 14 days.

This business problem shows strong reminisence to classic data science problems such as analysing churn for a company. In order to leverage this, an overview of the CRISP-DM process was achieved, this process was then followed and the steps were described along with the results for each phase. This work includes both experiments with different types of decision trees, different parameters for the models and using a dataset that have been tweaked to contain the same amount of retained and unretained customers.

With reference to the success criteria in Section 3.2, work have been done for all the entries on the list. A model have been created based on the available data, and a number of R scripts have been written that can recreate the model automatically and on request. Furthermore a number of tools have been created for manually validating the quality of the model using k-fold cross validation.

While Section 5 showed a number of different models, very little was learned about the life-cycle of a retained customer as opposed to an unretained customer. The most significant difference between the two is the number of logins and edits the user performs. Based on this data it looks like getting the customer to engage with the service regularly via logins and edits is what greatly increases the probability of the customer being retained.

In Section 5.2 I also outlined a rough design for using the model for an automated workflow, that could be used to automatically draw customer data from the database once every other week. The older customer data can then be used to determine if any of the new customers should be contacted to "activate" them. The script should place an email in an SQL based email queue to be sent out to the relevant customers.

In the same section I also briefly described an idea for a new dataset that might help the model become better at predicting the customers retention based on their behaviour. The dataset would also help Simplesite gain a better understanding of what separates retained and unretained

customers by increasing the accuracy of the model making it more representative for how customers use the service.

Furthermore, a number of different parameters for the model was tested and the results were documented for future reference. An attempt at manipulating the dataset to obtain greater predictive accuracy for retained users was also attempted, but ultimately was not useable due to lower mean accuracy.

The project is to serve as a prototype for how Simplesite can work with decision trees and data collected from their customers, with a proper version to be implemented by the companies own developers at a later time. The current datasets are created from data logs Simplesite already had available, in the future they will populate a dataset with event data extracted specifically for the purpose of analysing the user behaviour. When this dataset is finished this report and project can be used as a framework for how to work with the data.

References

[1] K. J. Bekkelund. Succeeding with freemium. *Innovation and Entrepreneurship, Specialization Project*, 2011.

[2] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[3] L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and regression trees. *The Wadsworth Statistics/Probability Series*, 1984.

[4] cvr.dk. Enhedsdata for simplesite aps, 2016. [`https://datacvr.virk.dk/data/visenhed?enhedstype=virksomhed&id=10079861`; accessed 11-January-2016].

[5] T. Hothorn, K. Hornik, and A. Zeileis. Unbiased recursive partitioning: A conditional inference framework. *Journal of Computational and Graphical statistics*, 15(3):651–674, 2006.

[6] K. Jensen. Crisp-dm process diagram, 2016. [`https://en.wikipedia.org/wiki/File:CRISP-DM_Process_Diagram.png` ; License: Creative Commons Attribution-Share Alike 3.0 Unported ; accessed 15-January-2016].

[7] F. Provost and T. Fawcett. *Data Science for Business: What you need to know about data mining and data-analytic thinking*. O'Reilly Media, Inc., 2013.

[8] N. Pujol. Freemium: attributes of an emerging business model. *Available at SSRN 1718663*, 2010.

[9] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[10] C. Shearer. The crisp-dm model: the new blueprint for data mining. *Journal of data warehousing*, 5(4):13–22, 2000.

A Dataset Bias Tables

	Reference	
Prediction	`FALSE`	`TRUE`
`FALSE`	85855	4488
`TRUE`	816	1583

Table 8: The predictions and reference values for running 5-fold cross validation on the training dataset with a maximum tree depth of 4.

	Reference	
Prediction	`FALSE`	`TRUE`
`FALSE`	85941	4570
`TRUE`	729	1501

Table 9: The predictions and reference values for running 5-fold cross validation on the training dataset with a maximum tree depth of 6.

	Reference	
Prediction	`FALSE`	`TRUE`
`FALSE`	85914	4507
`TRUE`	757	1564

Table 10: The predictions and reference values for running 5-fold cross validation on the training dataset with a maximum tree depth of 8.

	Reference	
Prediction	FALSE	TRUE
FALSE	5143	1297
TRUE	928	4774

Table 11: The predictions and reference values for running 5-fold cross validation on the training dataset with a maximum tree depth of 4. This is using a dataset with the same number of retained an unretained observations.

	Reference	
Prediction	FALSE	TRUE
FALSE	5042	1162
TRUE	1029	4909

Table 12: The predictions and reference values for running 5-fold cross validation on the training dataset with a maximum tree depth of 6. This is using a dataset with the same number of retained an unretained observations.

	Reference	
Prediction	FALSE	TRUE
FALSE	5053	1174
TRUE	1018	4897

Table 13: The predictions and reference values for running 5-fold cross validation on the training dataset with a maximum tree depth of 8. This is using a dataset with the same number of retained an unretained observations.

B Code

B.1 Tree Comparison

```
#install.packages('randomForest'); install.packages('party'); install.packages('rattle'); install

library(rattle)
library(rpart.plot)
library(RColorBrewer)
library(randomForest); library(rpart); library(party)
library(kimisc) # For dataset sampling.

source("tools.R")

dataset_train <- read_tsv("dataset_train.tsv")
dataset_test <- read_tsv("dataset_test.tsv")


################################################################################
# get same random seed each time for reproducibility
set.seed(1337)

dataset <- dataset_test

# Supress warnings while we run the tests. This is because we get a lot of closed
```

```
# sessions.
warn_level <- getOption("warn")
options(warn=-1)
print("=======================================")
print("ctree max depth 4:")
results <- ctree_k_fold_test(dataset, iscjretained ~ . , max_depth = 4)
k_fold_result_means(results)

print("rpart max depth 4:")
results <- rpart_k_fold_test(dataset, iscjretained ~ . , max_depth = 4)
k_fold_result_means(results)


print("=======================================")
print("ctree max depth 6:")
results <- ctree_k_fold_test(dataset, iscjretained ~ . , max_depth = 6)
k_fold_result_means(results)

print("rpart max depth 6:")
results <- rpart_k_fold_test(dataset, iscjretained ~ . , max_depth = 6)
k_fold_result_means(results)


print("=======================================")
print("ctree max depth 8:")
results <- ctree_k_fold_test(dataset, iscjretained ~ . , max_depth = 8)
k_fold_result_means(results)

print("rpart max depth 8:")
results <- rpart_k_fold_test(dataset, iscjretained ~ . , max_depth = 8)
k_fold_result_means(results)

# Restore warnings
options(warn=warn_level)
```

Listing 1: The code used to compare the different decision tree models. For the content of the `tools.R` file, see Figure 5. (../src/Report-TreeComparison.R)

B.2 Formula comparison

```
#install.packages('randomForest'); install.packages('party'); install.packages('rattle'); install

library(rattle)
library(rpart.plot)
library(RColorBrewer)
library(randomForest); library(rpart); library(party)
library(kimisc) # For dataset sampling.
library(foreach)

source("tools.R")

dataset_train <- read_tsv("dataset_train.tsv")
dataset_test <- read_tsv("dataset_test.tsv")
```

```

###############################################################################
# get same random seed each time for reproducibility
set.seed(1337)

#formulas <- generate_formulas(dataset_train, "iscjretained")
formulas <- c(iscjretained ~ . ,
              iscjretained ~ edits14 ,
              iscjretained ~ logins14 ,
              iscjretained ~ edits14 + logins14)


# Supress warnings while we run the tests. This is because we get a lot of
# closed sessions which spam inbetween iterations.
warn_level <- getOption("warn")
options(warn=-1)

num_formulas = length(formulas)

foreach(i = 1:num_formulas) %do% {
  print("=======================================")
  print("=======================================")
  print(paste("FORMULA: ", formulas[i]))

  for(j in c(4,6,8)) {
    print("=======================================")
    print(paste("DEPTH  : ", j))
    results <- ctree_k_fold_test(dataset_train, formulas[[i]] , max_depth = j)
    result <- k_fold_result_means(results)
    print(result)
    NULL
  }
  NULL
}

# Restore warnings
options(warn=warn_level)
```

Listing 2: The code used to compare the different formulas used for the model. For the content of the `tools.R` file, see Figure 5. (../src/Report-FormulaComparison.R)

B.3 Dataset Bias Test

```
#install.packages('randomForest'); install.packages('party'); install.packages('rattle'); install

library(rattle)
library(rpart.plot)
library(RColorBrewer)
library(randomForest); library(rpart); library(party)
library(kimisc) # For dataset sampling.
library(caret)

```

```r
source("tools.R")

dataset_train <- read_tsv("dataset_train.tsv")
dataset_test <- read_tsv("dataset_test.tsv")


################################################################################
# get same random seed each time for reproducibility
set.seed(1337)

# We wan't to try and see if the trees, look any different if we change the
# proportions of retained to non-retained customers.

# Seperate true and false values
all_true <- dataset_train[dataset_train$iscjretained == TRUE,]
all_false <- dataset_train[dataset_train$iscjretained == FALSE,]

# Find the largest
equality_size <- min(dim(all_true)[1], dim(all_false)[1])



dataset_equal <- rbind(sample.rows(all_true, size = equality_size),
                       sample.rows(all_false, size = equality_size))


################################################################################
# get same random seed each time for reproducibility
set.seed(1337)

# Disable warnings to avoid spam inside the loop.
warn_level <- getOption("warn")
options(warn=-1)

for(j in c(4,6,8)) {
  print("=======================================")
  print("=======================================")
  print(paste("DEPTH  : ", j))

  print("=======================================")
  print("Full training dataset:")
  results <- ctree_k_fold_test(dataset_train, iscjretained ~ . , max_depth = j)
  result <- k_fold_result_means(results)
  print(result)

  print("=======================================")
  print("Equal dataset:")
  results <- ctree_k_fold_test(dataset_equal, iscjretained ~ . , max_depth = j)
  result <- k_fold_result_means(results)
  print(result)
  NULL
}

```

```r
# Restore warnings
options(warn=warn_level)

################################################################################
# get same random seed each time for reproducibility
set.seed(1337)

test_ctree(iscjretained ~. , dataset_equal, dataset_test, max_depth = 4, TRUE )
test_ctree(iscjretained ~. , dataset_equal, dataset_test, max_depth = 6, TRUE )
test_ctree(iscjretained ~. , dataset_equal, dataset_test, max_depth = 8, TRUE )
```

Listing 3: The code used to test the impact of a dataset with the same number of retained an unretained customers. For the content of the `tools.R` file, see Figure 5. (../src/Report-DatasetExploration.R)

B.4 Results

```r
#install.packages('randomForest'); install.packages('party'); install.packages('rattle'); install

library(rattle)
library(rpart.plot)
library(RColorBrewer)
library(randomForest); library(rpart); library(party)
library(kimisc) # For dataset sampling.

source("tools.R")

dataset_train <- read_tsv("dataset_train.tsv")
dataset_test <- read_tsv("dataset_test.tsv")


################################################################################
# get same random seed each time for reproducibility
set.seed(1337)

test_ctree(iscjretained ~ . , dataset_train, dataset_test, max_depth = 4, TRUE )
test_ctree(iscjretained ~ . , dataset_train, dataset_test, max_depth = 6, TRUE )
test_ctree(iscjretained ~ . , dataset_train, dataset_test, max_depth = 8, TRUE )


################################################################################
# get same random seed each time for reproducibility
set.seed(1337)
dataset_train2 <- subset(dataset_train, TRUE, c(-logins14))
dataset_test2 <- subset(dataset_test, TRUE, c(-logins14))

test_ctree(iscjretained ~ . , dataset_train2, dataset_test2, max_depth = 4, TRUE )
test_ctree(iscjretained ~ . , dataset_train2, dataset_test2, max_depth = 6, TRUE )
test_ctree(iscjretained ~ . , dataset_train2, dataset_test2, max_depth = 8, TRUE )
```

Listing 4: The code used to produce the final results. For the content of the `tools.R` file, see Figure 5. (../src/Report-Results.R)

B.5 Utility Functions

```
#install.packages('foreach'); install.packages('doParallel'); install.packages('caret'); install.

library(foreach)
library(doParallel)
library(caret)
library(gregmisc)


################################################################################
######################### BEGIN UTILITY

read_tsv <- function(file) {
  dataset <- read.table(file,
                        sep="\t",
                        header=TRUE)

  dataset$isedit30m <- factor(dataset$isedit30m)
  dataset$isedit24h <- factor(dataset$isedit24h)
  dataset$isaddpage30m <- factor(dataset$isaddpage30m)
  dataset$isaddpage24h <- factor(dataset$isaddpage24h)
  dataset$isimgupload30m <- factor(dataset$isimgupload30m)
  dataset$isimgupload24h <- factor(dataset$isimgupload24h)
  dataset$iseditdesign30m <- factor(dataset$iseditdesign30m)
  dataset$iseditdesign24h <- factor(dataset$iseditdesign24h)
  dataset$iscjretained <- factor(dataset$iscjretained)
  dataset$isimgupload1d <- factor(dataset$isimgupload1d)
  dataset$iseditdesign1d <- factor(dataset$iseditdesign1d) # Was design edited within the first
  dataset$isaddpage1d <- factor(dataset$isaddpage1d)
  dataset$isedit1d <- factor(dataset$isedit1d)
  dataset
}


# Removes variables stored by foreach and registers back to the sequential backend.
unregister <- function(cl) {
  env <- foreach:::.foreachGlobals
  rm(list=ls(name=env), pos=env)
  stopCluster(cl)
  registerDoSEQ()
}

simple_plot <- function(tree, title = "Ctree") {
  plot(tree,
       title = "Ctree",
       type ="simple",           # no terminal plots
       inner_panel=node_inner(tree,
                              pval = FALSE, # no p-values
                              id = FALSE),  # no id of node

       terminal_panel=node_terminal(tree,
```

```
                                    abbreviate = TRUE,
                                    digits = 1,                    # few digits on numbers
                                    fill = c("white"),             # make box white not grey
                                    id = FALSE)
  )
}

# Generates a list of all permutations of formulas for a certain
# target variable/column name.
# Arguments:
#   dataset : The dataset to generate data from.
#   target  : A string with the target variables name.
generate_formulas <- function (dataset, target) {
  vars <- names(dataset)
  vars <- vars[! vars %in% target]
  indexes<-unique(apply(combinations(length(vars), length(vars), repeats=T), 1, unique))
  gen.form<-function(x) as.formula(paste(paste(target, ' ~ '),paste( vars[x],collapse='+')))
  formulas<-lapply(indexes, gen.form)
  formulas
}

######################### END UTILITY
###############################################################################



###############################################################################
######################### BEGIN K-FOLD VALIDATION

# Takes a list of outputs from the K-Fold tests below and gets the mean
# accuracy as well as the prediction / reference results.
k_fold_result_means <- function(input) {
  input_length <- length(input)
  accs <-
  foreach(i = 1:input_length) %do% {
    input[[i]][[1]]
  }
  mean_acc <- mean(unlist(accs))

  ttable <- input[[1]][[2]]
  foreach(i = 2:input_length) %do% {
    ttable = ttable + input[[i]][[2]]
  }
  mean_table = ttable / input_length
  ret <- list(mean_acc, mean_table)
  names(ret) <- c("MeanAccuracy", "MeanPredictions")
  ret
}

# Generates K sets of indices between 1 and Nobs (number of observations)
# Each set have a $train and $test attribute that contain the appropriate indices.
# From https://stackoverflow.com/questions/7402313/generate-sets-for-cross-validation-in-r
f_K_fold <- function(Nobs,K=5){
  rs <- runif(Nobs)
```

```
  id <- seq(Nobs)[order(rs)]
  k <- as.integer(Nobs*seq(1,K-1)/K)
  k <- matrix(c(0,rep(k,each=2),Nobs),ncol=2,byrow=TRUE)
  k[,1] <- k[,1]+1
  l <- lapply(seq.int(K),function(x,k,d)
    list(train=d[!(seq(d) %in% seq(k[x,1],k[x,2]))],
         test=d[seq(k[x,1],k[x,2])]),k=k,d=id)
  return(l)
}

# Uses K-fold cross validation to determine the accuracy of a given formula
# using ctrees.
# Arguments:
#   dataset     - The dataset to partition and validate against.
#   formula     - Formula to use for ctree construction.
#   K           - how many folds to use for K-fold cross validation. Default = 5.
#   max_depth   - The maximum depth of the ctree. Default = 4
#   num_threads - Maximum number of threads for parallelisation. Default = Number of physical cor
ctree_k_fold_test <- function(dataset, formula ,K = 5, max_depth = 4, num_threads = parallel:::de
  cl <- makeCluster(num_threads)
  registerDoParallel(cl)

  nobs <- dim(dataset)[1]
  idx_sets <- f_K_fold(nobs, K)

  accs <-
  foreach(i = 1:length(idx_sets)) %dopar% {
    library(party) # When using %dopar% you will need to load the libraries in each thread.
    library(caret)
    idx_set <- idx_sets[[i]]
    test_idx <- idx_set$test
    train_idx <- idx_set$train

    dataset_test <- dataset[test_idx,]
    dataset_train <- dataset[train_idx,]

    fit <- ctree(formula ,
                 data=dataset_train ,
                 controls = ctree_control(maxdepth = max_depth))
    predictions <- predict(fit, dataset_test)
    values <- dataset_test$iscjretained
    mat <- confusionMatrix(predictions, values, positive = 'TRUE')
    list(mat$overall[1], mat$table)
  }
  # Go back to sequential backend and return the data.
  unregister(cl)
  accs
}


# Uses K-fold cross validation to determine the accuracy of a given formula
# using decision trees from rpart.
# Arguments:
```

```
158 #   dataset     - The dataset to partition and validate against.
159 #   formula     - Formula to use for ctree construction.
160 #   K           - how many folds to use for K-fold cross validation. Default = 5.
161 #   max_depth   - The maximum depth of the ctree. Default = 4
162 #   num_threads - Maximum number of threads for parallelisation. Default = Number of physical cor
163 rpart_k_fold_test <- function(dataset, formula ,K = 5, max_depth = 4, num_threads = parallel:::de
164   cl <- makeCluster(num_threads)
165   registerDoParallel(cl)
166
167   nobs <- dim(dataset)[1]
168   idx_sets <- f_K_fold(nobs, K)
169
170   accs <-
171     foreach(i = 1:length(idx_sets)) %dopar% {
172       library(rpart) # When using %dopar% you will need to load the libraries in each thread.
173       library(caret)
174       idx_set <- idx_sets[[i]]
175       test_idx <- idx_set$test
176       train_idx <- idx_set$train
177
178       dataset_test <- dataset[test_idx,]
179       dataset_train <- dataset[train_idx,]
180
181       fit <- rpart(formula ,
182                    data=dataset_train ,
183                    method="class",
184                    control = rpart.control(maxdepth = max_depth))
185       predictions <- predict(fit, dataset_test, type = "class")
186       values <- dataset_test$iscjretained
187       mat <- confusionMatrix(predictions, values, positive = 'TRUE')
188       list(mat$overall[1], mat$table)
189     }
190   # Go back to sequential backend and return the data.
191   unregister(cl)
192   accs
193 }
194
195 ######################### END K-FOLD VALIDATION
196 ###############################################################################
197
198
199
200 ###############################################################################
201 ######################### BEGIN TREE TEST / WORK
202
203 # Will train a ctree model using the given formula and training dataset,
204 # test it with the test set and report accuracy.
205 # Arguments:
206 #   formula       - Formula to use for model.
207 #   dataset_train - Training dataset.
208 #   dataset_test  - Testing dataset, is expected to also contain target column iscjretained.
209 #   max_depth     - The maximum depth of the ctree, to let the algorithm decide, use a big number
210 test_ctree <- function(formula, dataset_train, dataset_test, max_depth, plot_tree = FALSE) {
```

```
211   fit <- ctree(formula ,
212                data=dataset_train ,
213                controls = ctree_control(maxdepth = max_depth))
214   if (plot_tree) { simple_plot(fit) }
215   predictions <- predict(fit, dataset_test)
216   values <- dataset_test$iscjretained
217   mat <- confusionMatrix(predictions, values, positive = 'TRUE')
218   list(mat$overall[1], mat$table)
219 }
220
221 ######################## END TREE TEST / WORK
222 ###############################################################################
```

Listing 5: Utility functions used throughout the different code files. (../src/tools.R)