Analysing clickstream data from Danskebank.dk

Project in Practice

June 11, 2018

Andreas Borgstad (pmh477), Christoffer Thrysøe (dfv107), Xuwen Zhang (dlv618)

Main supervisor: Fabian Gieseke, co-supervisor: Isabelle Augenstein

Contents

1 Abstract

We asses the usefulness of Celebrus, a tool used to capture customer behaviour through clickstream data on `danskebank.dk`. We provide an outline of the data produced, and explain how the data can be processed end-to-end by means of a processing pipeline, using Spark on Hadoop for feature generation, and Python/Sklearn/Keras for modeling. We propose three different ways of representing a click-path: Bag of Words, Word2Vec, and a deep learning technique inspired by Yoon Kim in [1]. We use RandomForest and a multilayer perceptron model to evaluate our representations, through 3 practical tasks. In the first two tasks, we predict the age of visitors, using a custom loss function. In the last task we predict whether visitors will click *"Become a customer"*. The best results was found using the deep learning approach presented.

2 Introduction

The home page of Danske Banke: `danskebank.dk` is a vital online platform to Danske bank, which on a daily basis is used by more than 400.000 visitors, ranging from customers to potential customers. From here, visitors can access various information about the bank, book a meeting, become a customer and more. Existing customers also have the option of logging on to their online bank, which offers the basic banking services also supported at the local bank branch, such as paying bills, viewing account balance, transferring money etc. In the past decade, banks have taken advantage of the technological advancements in society and moved a great deal of their services from local branches to online- and mobile services. This has resulted in a vast decline in the number of physical bank branches, and thereby moved its customers to the online platforms. Thus these online platforms have become the banks main tool for interacting with customers and potential customers.

Figure 1: Snippet showing the frontpage of `danskebank.dk`

The online platforms are where the bank must meet its customers demands and acquire new customers, thus the importance of these platforms to the bank can not be overstated and must at all time be improved to ensure the goals of the bank. Online platforms and the digitalization of banking systems, generates an immense amount of data, which has transformed the banking world. By analyzing the large amount of user data, it is possible for banks to deploy machine learning algorithms, which allows the learning of customer tendencies. These can then be used to provide tailored services to its customers, and thereby make the seemingly generic online platforms customer centric. These customer tendencies or interests can be based on a wide variety of information. For example the customers age will, in most cases, indicate whether a customer could be interested in buying a house or not, but other more subtle features, such as the key stroke frequency or the customers browsing history on the banks website in the past month, may identify a certain group of customers. The big data approach is applicable for a wide array of tasks within the bank, for example risk assessment, predicting if a customer is leaving the bank, predicting if the anomalous behaviour of some customers indicate fraudulent transactions, and many more. Big data provides valuable insights into customer behaviour and is now an extremely valuable resource in helping the bank maximizing its profit and customer satisfaction.

As one of the biggest banks in the Nordic, operating in Scandinavia, UK and Luxembourg, Danske Bank has started taking a keen interest in big data, and is dedicated to advanced analytics of big data and the continuous development of prediction models and data acquisition. A new tool, called Celebrus, has been employed at the bank, to help support this dedication. The Celebrus tool captures visitor behaviour on the Danske Bank home page, by logging user-generated events, such as clicks, text field submissions, and more. The tool also captures meta-data of the visitors interaction with the web-page, such as browser, operating system of the used device and more. The deployment of the Celebrus tool is in correlation with the banks

interest in data driven decision making.

As the tool is new, it is yet unknown what applications the data will have, if any, but as the data is widely applicable it is expected that the data will be used by several departments of the bank for a variety of different purposes. One area that the bank is already committed to improving, through the Celebrus data, is becoming more proactive for its customers and potential customers. That is for example if a visitor is exhibiting the behaviour of similar visitors who were interested in applying for a loan, instead of waiting for this customer to apply for a loan, the bank could proactively offer a loan to the visitor instead. The bank is expecting to greatly increase it's revenue, by making the online platforms more customer-centric and introducing pro-activity.

3 Project Goals

The goal of the project, and this report, is to explore the Celebrus data in detail, both by carefully inspecting the data and by using the data to solve selected tasks, such that the report can be used by the Advanced Analytics & Architecture (AAA) department of Danske Bank, when working with the Celebrus data. The findings, which we will make throughout the project and the obtained experimental results, will then be presented in this report. The description will target readers who are unfamiliar with the data, but also readers with knowledge of the data, who wishes to incorporate or use the data for modeling. The description will cover how the data is obtained and structured, what the data consists of, i.e. the information captured by the Celebrus tool, and a processing pipeline, which shows how the data can be processed.

To assess the data and discover potential shortcomings, regarding the data, we wish to examine whether the data can be used to abstract certain visitor behaviours and traits, which could be of importance to the bank. We will do this by transforming the data into features and model the given behaviour through machine learning methods. Here we will model several visitor traits, which we selected, and discuss what features from the Celebrus data was used and which approach gave the best results. These visitor traits, are visitor-level behaviours, i.e. we do not wish to model behaviours across visits. We will also show how the data was prepared and transformed into relevant features, using the available frameworks, also used by the data scientists at the bank. Here we will go into depth about how we chose to represent the data and which representation gave the best results. This is intended to guide the data scientists at the bank, who wishes to use the data for modeling, by showing a processing pipeline, which can be directly employed in the banks ecosystem, but also for us to get familiar with these big data tools.

The bank expects to deploy the Celebrus data in a wide range of models in the bank and in different branches. Therefore as the data is not being used in the bank yet, the goal of this project is not to solve an existing task in the bank, through the Celebrus data, but rather explore the data and lay the foundations of models to come, reducing the preliminary work needed to be done by the data scientists at the bank, when working with the Celebrus data. What this project also won't do is to produce a production-ready end-to-end solution within Spark, that can be directly deployed to solve the tasks. We will not harden our processing pipeline for production, but keep it experimental and rather describe it in detail to give an idea of how to approach the Celebrus data. We will also restrict our attention visitors using the Danish version of the webpage.

4 Technical overview

In this section we will give a short walk-through of the work setup we had at the bank, and briefly outline the technical frameworks used.

4.1 Platform

The current goal of the Advanced Analytics & Architecture (AAA) team in Danske Bank, which this project is involved with, is to serve and maintain a data lake, and perform data science related tasks - that is, to have a huge range of datasets, gathered across the departments of the bank, from source systems to be used for analytics purpose. In the current setup in Danske Bank, an enterprise Hadoop distribution is provided by HortonWorks together with a tool stack, primarily focused on ETL, governance, and quality assurance. Spark is also included in this solution.

Technical specifications for Hadoop can be seen summarized in below table:

Nodes	29
Storage	1.5 PB
CPU Cores	1044
Memory	4.9 TB

When referring to Hadoop, we refer to the base Apache Hadoop framework, which consists of the Hadoop common library, Hadoop Distributed File System (HDFS) for disk storage, Yet Another Resource Negotiator (YARN) for resource negotiation, and MapReduce which is for processing big data. The components will briefly be described in the sections below.

We also have a GPU server at our disposal, with the following specifications. It will later be referenced as the *Tesla server*:

Storage	5 TB
Memory	500 GB
Graphics cards	2 X Tesla M40

4.2 MapReduce

MapReduce consists of three logical steps: Map, Shuffle and Reduce. The first step computes for each mapper, a key and a value per data item, and returns a tuple of key/value pairs, by performing a function on each of the data items. These are shuffled, meaning that the corresponding pairs goes to the same reducer. The reducer returns for each key, a value (or list of values). The output of the shuffling procedure is written to disk, and the same is the output of the reduce procedure. Since writing to disk is an expensive procedure, it is something one wishes to avoid, when executing algorithms that depends on a relatively large number of iterations. This could for example be algorithms that relies on gradient descent - this rules out neural networks, since this would require a write to disk, for each back-propagation.

4.3 HDFS

Hadoop was built to be easily scalable, allowing users to add (commodity) hardware to a cluster, thereby increasing its computation power, and storage. While increasing the number of machines, it is not a question of if a machine will fail, but when. Hadoop was designed with this in mind[1]. Therefore, HDFS facilitates

[1] https://research.google.com/people/jeff/stanford-295-talk.pdf

data replication - on the Danske Bank setup cluster, the replication factor is 3 - i.e. if a file is written to HDFS, it is written to three separate nodes in the cluster.

4.4 YARN

This library is used to negotiate resources for processing. A Hadoop Cluster typically contains a single NameNode, and several DataNodes. If a user requests a MapReduce execution, the job goes through the NameNode, and is hereafter delegated to a single DataNode which will be denoted the Application Master - the Application Master will negotiate computational resources (in the from of worker nodes) with the NameNode, and is responsible for serving the client the result, once it is calculated. Roughly, in the early versions of Hadoop, the NameNode acted as the Application Master, for all incoming jobs - for large clusters, this created a bottleneck, where the NameNode would become congested.

4.5 Spark

Spark is a framework, which is used to perform data processing on a cluster, through a rich high-level API available in popular programming languages such as Java and Python. The motivation behind Spark is to distribute the data to be processed, across a set of nodes on the cluster, and then process these datasets in parallel. Unlike MapReduce frameworks, Spark drastically speeds up processing time by performing most of the required computations in-memory, which makes it suitable for performing real time analysis. Spark does so by performing all computations on resilient distributed datasets (RDD). RDD's are typed collections of partitioned data, which resides in the memory of the nodes, distributed on the cluster. RDD's support two types of actions: Transformations and actions. Spark is lazy evaluated and awaits for an action to be called before performing any distributed transformations and actions on the cluster. This allows Spark to optimize the chain of transformations to be performed on the data, before executing the action. For our task, we will mostly use the data abstraction `Dataframe`, which functions as a collection of data, structured into named columns, similar to a database table. Spark provides a library called SparkSQL for performing efficient queries on Dataframes. Spark exhibits a master/worker architecture, as shown in figure 2, in which there is a master (or driver), who keeps track of the workers nodes, which executes the operations.

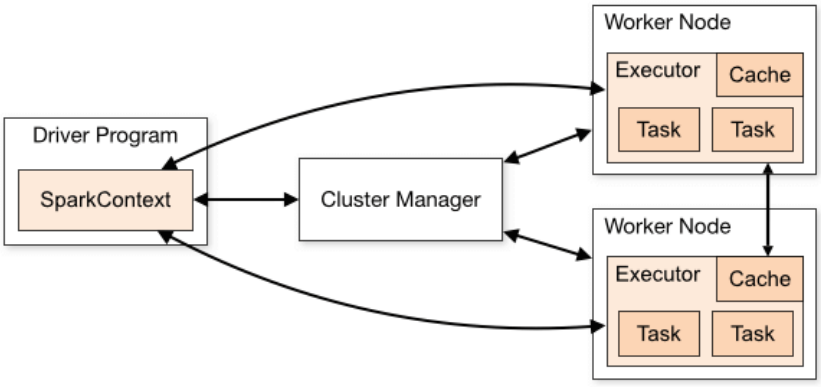

Figure 2: Overview of sparks master/worker architecture. Picture taken from `https://spark.apache.org/docs/latest/cluster-overview.html`

The driver hosts the SparkContext, which is responsible for establishing an environment on the cluster so that the spark application can be run. The driver is responsible for orchestrating the execution of the given task by allocating executors on the worker nodes to execute the tasks, with permission from the cluster manager. A worker node usually holds multiple executors for different tasks to be done, each allocating resources such as memory to hold RDD's.

4.6 Our setup

In the duration of the project, we were using a Lenovo laptop each, provided by the bank. The setup can be seen in figure 3.

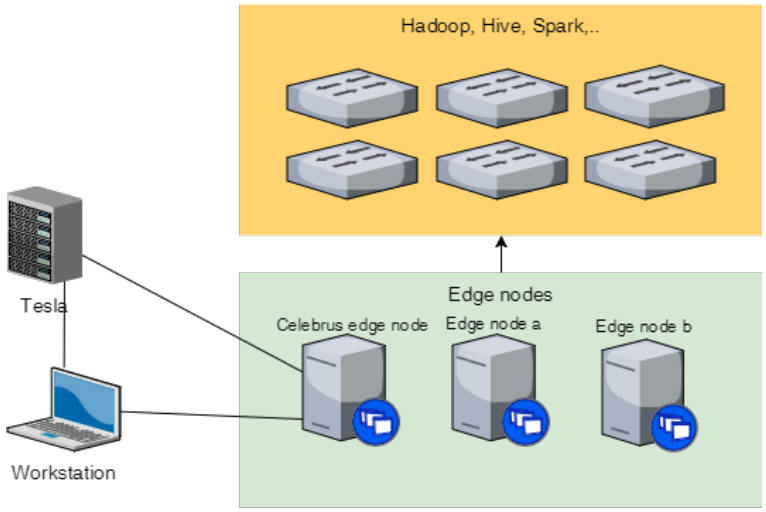

Figure 3: Our work setup at Danske Bank.

The edge nodes graphed in the figure, are virtual servers which are ordered and added to a network enclave connected to the cluster, when a new project is initiated in AAA. These edge nodes provides access to the banks big data technical stack, and are themselves a RedHat Linux distribution. The access to Hadoop is provided through client applications, that can communicate with Hadoop. The edge nodes also contains a Jupyter notebook for Spark usage.
The Celebrus edge node is a virtual server, which we accessed by means of `ssh` through putty[2].

Besides the edge node, we also got a database in Hive: `34np_project_celebrus_ku`. The Celebrus data resides in a production table in Hive, and therefore cannot be accessed directly through Spark, which is where we process it, due to security reasons. Therefore, we have to copy the Celebrus data relation, to `34np_project_celebrus_ku` where we can access it through Spark.

The Tesla server is not in the edge node enclave and therefore not connected to Hadoop. This practically means, that creating models on it requires moving data to it - this is done through the Linux application SCP.

Jupyter Notebook has been used for generating figures, and doing initial exploration of data. The pipeline and models described in this report, have been programmed in Python.

[2] `https://www.putty.org/`

5 Celebrus

Celebrus is a tool, which captures visitor behaviours on the `danskebank` web-page, and stores this information in a relational database. The behaviour of a visitor is logged whenever the visitor triggers an event on the given web-page. An event can be a click on the page, insertion in a text field and more. The current Celebrus setup is restricted to click and text field insertion events, Celebrus can however be configured to use user-defined triggers, for example when testing a new product.

In this section we will provide a detailed overview of the Celebrus tables, which are relevant for our tasks. The columns of the tables we found to be most useful is here presented. A complete description of the tables, as provided by Celebrus, can be found in the Appendix in section 14.1. We will refer to a visit on the webpage as a session. A session starts when a visitor first accesses a webpage on the domain and ends when the visitor closes the browser or is idle for more than five minutes.

5.1 Table description - with examples

The Celebrus relational database is stored on Hive, and holds 51 tables, where 24 of the tables currently contains data. The other 28 tables are currently empty, and requires configuration to be set up. When an event is triggered, a row is inserted into the Hive Celebrus database. In this section we will describe and provide examples, for the tables that we used to extract features from. The tables we will outline are:

- Visitor
- Page
- Sessionstart
- Click

Each table has `sessionnumber` as primary key, which is a 5-20 digit integer that uniquely identifies a visitor session.

5.1.1 Visitor

The visitor table is used to identify the visitor of a session. A row is inserted in this table, whenever a new session starts, without the ebanking agreement number. Once the visitor logs in, a new row is inserted in visitor table, where the `profileuiid` column is filled with the customers ebanking agreement number. The ebanking agreement number is an alphanumeric string, used within the bank to uniquely identify a customer.

Table 1: **Example of visitor table**

sessionnumber	profileuiid	eventtimestamp	idsequencenumber
3125737	None	1513181757330	1
3125737	58363F	1513182331350	2
5147632	None	1513182335232	1

Table 1 shows an example of the visitor table. For sessionnumber 3125737, two entries in the table exists: one without a profileuiid and one with. Thus the visitor entered the webpage, which triggered an entry in the Visitor table without the profileuiid. When the customer logged in, a new entry was made in the table, which has the profileuiid filled out. Session 5147632 only has a single entry filled in the table, without a profileuiid, thus throughout the session the visitor did not login.

5.1.2 Page

An entry in the page table is generated each time a page load occurs, thus there is as many entries for a session as the number of clicks made by the corresponding visitor, which resulted in a pageload. A pageload happens, whenever data is sent to the visitor from the webserver, updating the view of the page. The updated entries are as followed:

- **pagetitle**: Value embedded by the `<title>` HTML tag.
- **profileuiid**: Unique identifier for each customer, such that when the customer logs in, a mapping is created between the sessionnumber and profileuiid. Note that profileuiid is a name dubbed by Celebrus. Internally, this value is called the ebanking agreement number.
- **eventtimestamp**: Unix timestamp in milliseconds associated with the time for which the page load occurred.
- **pagesequenceinsession**: Sequence number of the page within the session, i.e. the index, identifying the order for which the given pageload occurred in the session.
- **pagelocationdomain**: The base domain at which the pageload occurred, e.g. *"danskebank.dk"* and *"netbank2.danskebank.dk"*
- **pagelocation**: The full URL of the current page at which the page load occurred.

Table 2 shows the page table with data from a single session stored.

Table 2: **Example of page table**

sessionnumber	pagetitle	eventtimestamp	profileuiid
2033877	Du skal købe bolig	1513540229861	None
2033877	Låneguide - boliglån	1513540258176	None
2033877	Find dit boliglån	1513540293372	None
2033877	FlexLife	1513540341491	None
2033877	Find dit boliglån	1513540386058	None
2033877	FlexLån-T	1513540428271	None
2033877	Logon NemId	1513540538878	None
2033877	Danske Netbank	1513540541431	58363F
pagesequenceinsession	**pagelocationdomain**	**pagelocation**	
1	danskebank.dk	/privat/mit-liv/bolig/koebe-bolig	
2	danskebank.dk	/privat/mit-liv/bolig/koebe-bolig/laaneguide	
3	danskebank.dk	/privat/produkter/laan/bolig-laan	
4	danskebank.dk	/privat/produkter/laan/bolig-laan/flexlife	
5	danskebank.dk	/privat/produkter/laan/bolig-laan	
6	danskebank.dk	/privat/produkter/laan/bolig-laan/flexlaan-t	
7	netbank2.danskebank.dk	/pub/logon/logon.aspx?ss=JI	
8	netbank2.danskebank.dk	/REB/SecureMes/SecureMessageCreate.aspx?	

The visitor, outlined in table 2 performs the following actions on the webpage:

1. The visitor clicks on information about buying property.
2. The visitor then looks at a loan guide.
3. The visitor then goes to products to look at various available loans.

4. The visitor clicks on a specific loan product called "*FlexLife*".
5. The visitor then goes back to the overview of the available loan products.
6. The visitor clicks on a different loan product called "*FlexLoan-T*".
7. The visitor clicks on the login button.
8. The visitor is now logged in on the netbank.

We see from table 2 that once the visitor is logged in, the profileuiid is listed. From this table we sampled the URL's, i.e. the click paths, of each session by extracting the column `pagelocation` .

5.1.3 Sessionstart

An entry in the sessionsstart table is generated once a new session is initialized. The sessionsstart table holds only one row per session and lists information about the given session. Some of the information stored includes:

- **deviceIPAddress**: The ip-address of the device used to visit the web-page.
- **DeviceSystemType**: The type of device used to visit the web-page, e.g. personal computer, mobile device, game platform device etc.
- **DeviceBrowserName**: The name of the browser used to visit the web-page.
- **DeviceBrowserVersion**: The version of the browser, if available.
- **DevicePlatformName**: The name of the operating system of the device used to visit the web-page.

Table 3 shows the sessionsstart table for different sessions.

Table 3: **Example of SessionStart table**

sessionnumber	**deviceipaddress**	**devicesystemtype**
2035477	83.24.215.69	PC
33512504	85.72.161.129	Mobile/PDA
33554637	2.126.254.6	PC
33512532	86.191.12.14	Mobile/PDA
33581560	81.133.84.113	PC
devicebrowsername	**devicebrowserversion**	**deviceplatformname**
MS Internet Explorer	11.0	Windows 7
Opera	None	Android
Google Chrome	64.0.3282.186	Windows 10
Safari:iPad	604.1	iOS
Safari	604.5.6	MacOS X

5.1.4 Click

The click table stores clicks the visitors makes on the website. The columns in the table each describe HTML object attributes. Once the visitor clicks on a page object, a row is inserted in the click table, where the columns are filled with the corresponding HTML attributes of the page object. Because the HTML objects mostly contain only a few attributes, the rows in the table are sparse, i.e. a lot of the columns are set to None as the given attribute was not listed for the clicked object. An example of the click table is shown in table 4, which shows the entries for the same visitor as in the page table.

Table 4: **Example of click table**

sessionnumber	objectalttext	objecttype	objectclass
2033877	Købe bolig	OTHER	None
2033877	Læs mere	OTHER	None
2033877	Se alle boliglån	LINK	1 icon arrow-right
2033877	use xmlns	OTHER	category-toggle-button button blue
2033877	Læs mere	OTHER	None
2033877	use xmlns	OTHER	category-toggle-button button blue
2033877	Læs mere	OTHER	None
2033877	None	OTHER	icon add large
2033877	Log på	LINK	button show-for-medium
2033877	Send besked	LINK	arrow_link
objecthref			
privat/mit-liv/bolig/koebe-bolig			
privat/mit-liv/bolig/koebe-bolig/laaneguide			
privat/produkter/laan/bolig-laan			
javascript:void(0)			
privat/produkter/laan/bolig-laan/flexlife			
javascript:void(0)			
privat/produkter/laan/bolig-laan/flexlaan-t			
javascript:void(0)			
site-ressources/login-page/logon-fallback			
javascript:top.logo			

As evident from the table a large amount of the entries are set to None. We also see that the table has more entries than the page table. This is due to some of the clicks, made by the visitor, not triggering a pageload, for example the visitor expands the list of loan products, which is loaded dynamically on the webpage using Javascript, and thereby not being registered as a pageload. It is difficult to decipher what the visitor actually clicked on as the information is spread across the different attributes. For example for a Javascript load, it would be necessary to look at the objectclass of the clicked HTML object to uniquely identify what the visitor clicked on. Ultimately we decided not to use the click table to uniquely identify customer clicks, but rather the page table, as this always identifies what page the visitor is browsing. The drawbacks of this is that clicks, which triggers a Javascript event is not uniquely identifiable as the table only captures the current page URL for which the event occurred. This practically means that the page table will have consecutive rows of the same URL. This could have been mitigated by combining the page url and different click table attributes.

5.2 Other tables

A complete overview of all the tables avaliable in the Celebrus data relation can be seen in appendix 14.2, together with their row counts. The tables containing the words "field" and "form", captures data about visitor interaction with forms or fields on a website. This could for example be used to capture search terms entered into the search field, at the frontpage of danskebank.dk. The tables containing the word "campagin", has information about visitors who accessed danskebank.dk by an online advertisement on another site, or advertisements placed on danskebank.dk, to announce new products or offerings.

5.3 High-level visualization

High level information on visitor behaviour has been gathered and plotted, to develop an intuition about the data.

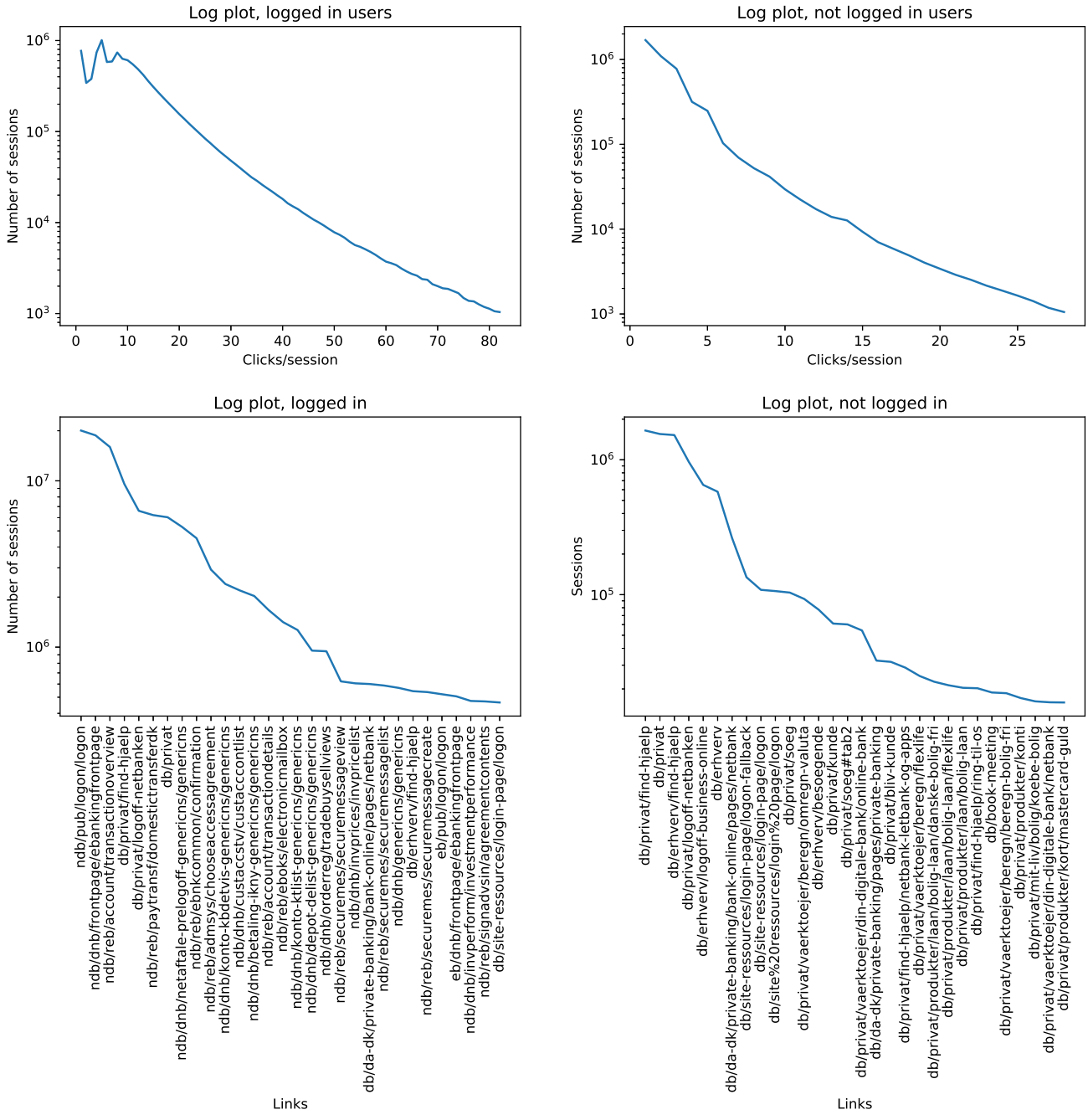

Figure 4: Left column shows statistics for logged in visitors, and right column shows statistics for not-logged in visitors. db is an abbreviation for danskebank, and ndb is an abbreviation for netbank2.danskebank.dk

The X-axis of figure Figure 4 shows the number of clicks per session, and the Y-axis shows the number of sessions, that has a x amounts of clicks. Generally visitors who logged in have more clicks, than non logged in visitors. For logged in visitors, the most frequent numer of clicks per session is 5 clicks per sessions. For

not logged in visitors, the most frequent number of clicks per session is 1 click per session. For both groups, the frequency in number of clicks per session decreases as the number of clicks per session increases.
The second row shows the most popular pages, for logged in visitors (left), and not logged in visitors(right). Parts of the URL's has been stripped away for readability.

6 Selected Tasks

In this section, the three selected tasks will be outlined. For each task, we will introduce the problem and why we believe that the task is relevant to the bank. We will also discuss what features, in the Celebrus data, we believe will be helpful when solving the task.

6.1 Task 1. Predicting ages of visitors

For this task, we wish to predict the ages of visitors to `danskebank.dk`, based solely on the Celebrus data. We wish to do so because knowing the age of a visitor can to a great extend infer the potential interests of said visitor. For example a visitor, older than thirty years old will most likely not be interested in student loan information, and likewise, an eighteen year old visitor will most likely not be interested in information regarding his/hers pension. Visitors to `danskebank.dk` can get information about becoming a customer, by clicking on the top-menu bar link: *"Become a customer"*. This link takes the visitor to a page, where the visitor can search information on becoming a customer. Upon clicking the link, the visitor is met by a questionaire, which requests the age, profession and living situation of the visitor. This questionnaire is shown in figure 5.

Figure 5: Questionaire, shown on the "*become a customer*" page, requesting age, profession and living situation.

Filling in the above questionnaire, triggers a list of sign-up offers, that directly targets what the user has put in the questionnaire. To receive these offers, it requires that the visitor goes to the *"Become a customer"* page and fills out the questionnaire. This also means that a great deal of visitors, potentially interested in these offers, may not sign up to become a customer, as they do not know these offers exists. If the bank, however could infer the age of the visitor, based on the behaviour of the visitor, then relevant offers could be shown proactively to the visitor, without requiring the visitor to fill in the questionnaire. We believe that it is possible to extract features and use these to predict the age of the visitor, using the Celebrus data. We believe that the following data, from the Celebrus data, will help us solve the task:

- **The sequence of visited pages**. Visitors of different ages will most likely have different interests or different objectives on the webpage. Visits to certain categories of pages indicate the interest of the visitor and thus potentially the age of the visitor. To verify this, we have taken two age groups: $[18-27]$ and $[60-84]$. For both age group we take the top 40 most frequent webpages and take the set difference for each group. This is done because a large amount of the most frequent webpages are common for both age groups, for example the main- and login page. Table 5 shows the top-three URL's for each age group, which are not included in the other age group. To make the URL's more readable, the Danskebank domain has been stripped.

Table 5: Most frequent links for age groups [18-27] and [60-84], which is not frequent in the other age group top frequent clicks.

[18-27]	[60-84]
/dnb/danskenetopsparing_ung_bestil	/dnb/invagreem/invagreementoverviewpage
/dnb/mastercard_direct_bestil	/dnb/invperform/investmentperformance
/dnb/youngstudent	/dnb/invagreem/invagreementcostspage

As evident from the table, the listed URL's corresponds well with the age groups. For the age group $[18-27]$, the listed URL's concerns with ordering a youth savings account, ordering a Mastercard and information regrading a youth savings account, while for the age group $[60-84]$ all the URL's concerns investment links, such as overview of investments. Thus the two age groups have different interests as expected.

- **Time between clicks**: Click frequency can potentially indicate if the visitor is more experienced, when using a computer. This experience can reflect the age of the visitor, as computers are more broadly accessible than they were decades ago. Figure 6 shows the time between clicks.

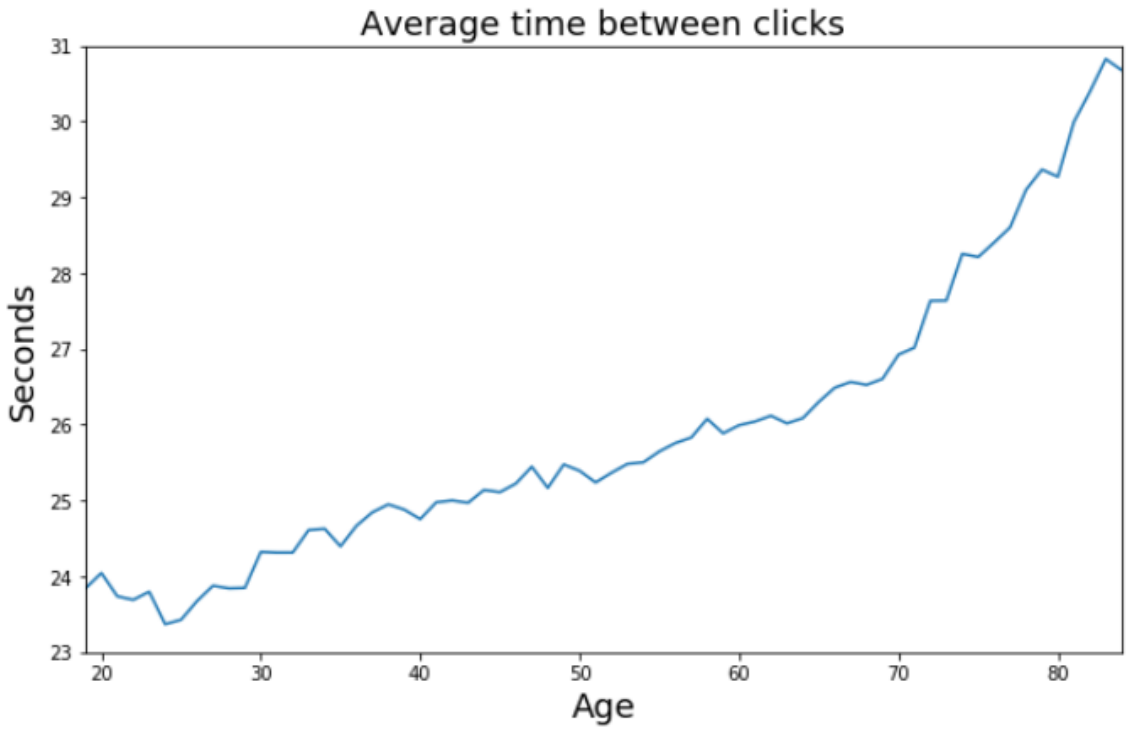

Figure 6: The average time between clicks in seconds, as a function of ages.

From the figure, it can bee seen that there is a slight tendency between age and the average time between clicks, although the change in time (seconds) is small.

- **Useragents**: Operating system and browser may indicate the age of the visitor. Figure 7 shows the distribution of used operating system of each age, normalized with respect to the frequency of each age.

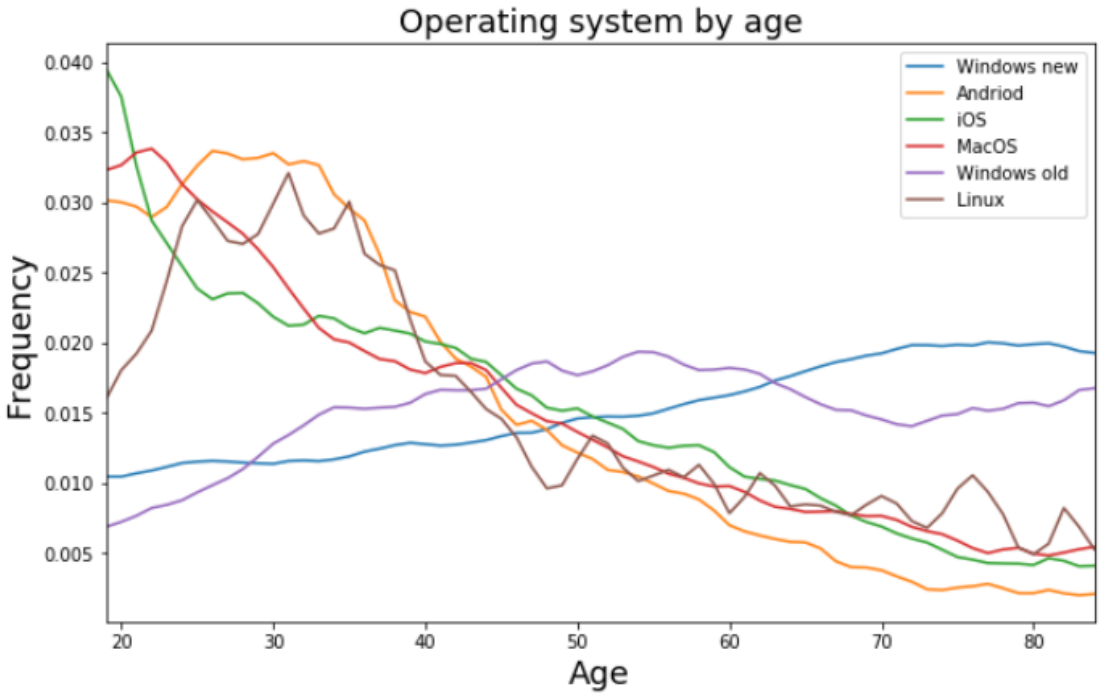

Figure 7: The most used operating system for each age, normalized with the respect to the frequency of the age.

Two operating systems are clearly more frequent for ages greater than 60: Windows old (Windows 7 and older) and Windows new (Windows 8, 10). Likewise we see that IOS and MacOS is frequently used in the age interval [19-24], while Android and Linux is more frequent in the age interval [25-35]. Thus there is a connection between the age and the used operating system.

- **Time of visit** The time of day/week that the visitor visited the webpage may indicate the age of the visitor, e.g. students may have more flexible hours than people with a full time job, which may reflect on when they use their online bank. Figure 8 shows a heat map of the visit time for two different age intervals. The weekday is displayed on the x-axis and the hour of login is displayed on the y-axis.

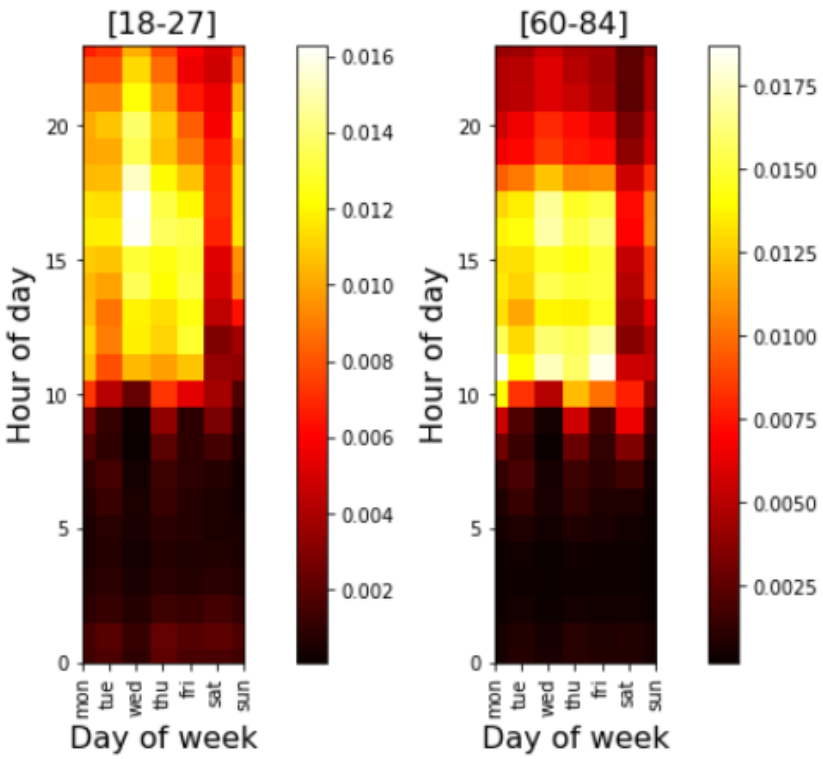

Figure 8: The distribution of login times for age groups [18-27] and [60-84]

As evident from the figure, the two age groups have somewhat different visiting patterns. The age interval [18-27] produces more visits during the evening at mid-week, whereas visitors in the age interval [60-85] are using the website less frequent in the evening.

Thus, from the above observations, we believe that the task is solvable through the Celebrus data. We will treat the ages of visitors as a continuous variable, and thereby formulate the problem as a regression task. For the task, we will only consider visitors between the ages of 18 to 84 years old. One challenge for this task is that the dataset is highly unbalanced. Figure 9 shows the frequency of visitors for each age. As evident from the figure, the majority of the visitors is in the age interval 45 - 75.

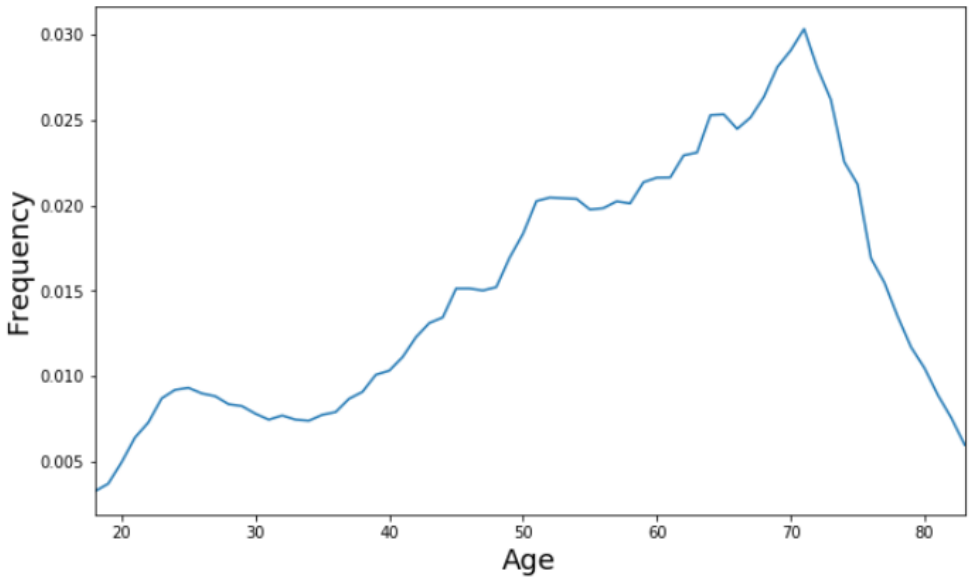

Figure 9: Frequency of each age in the Celebrus data.

6.2 Task 2. Predicting age groups

This task is coupled with the first task in the sense that it consist of predicting the ages of visitors. As shown in task 1, tailored offers are shown to visitors who fills out the questionaire, shown in figure 5. Figure 10 shows an example, where the visitor has put in the age 25. This triggers a specific offer for visitors in the age group 18-27 years old, which includes a free account, youth financial counselling, bike insurance and offers which is targeted this age group.

Figure 10: A list of sign-up offers for visitors in between the ages 18 and 27.

As mentioned, if the bank could infer that the visitor is within the age group 18-27 years old, then this offer could proactively be shown to the visitor, to ensure that the visitor receives the offer . Thus for the second task we wish to perform binary classification on whether visitors belong to age group $[18-27]$ or $[28-84]$. We believe that we can use the same features as for task 1 as the this task is almost identical. Looking at the distribution of visitors among the ages in the dataset, shown in figure 9, it is clear that this task also has an unbalanced dataset as the age group 18-27 is under represented in the data. The frequencies of the two age groups is listed in table 6.

Table 6: Frequencies of the two age groups in the dataset.

[18-27]	[28-84]
0.0718	0.9281

6.3 Task 3. Predicting "become a customer" click

For our third and final task, we wish to predict whether visitors click on the "*become a customer*" link. Thus the task is a binary classification task, where we want to predict if a customer visits the page or not. We will generate the targets by simply checking whether or not the link is in the visitors click path or not. If a session contains clicks after clicking on the "*become a customer*" link, these clicks are removed, together with the "*become a customer*" link. The "*become a customer*" link is accessible at the top menu of the homepage of Danskebank, shown in figure 11 and as mentioned in the previous task, this link redirects the visitor to information on how to sign up to be a customer and offers various ways of doing so.

Figure 11: Top menu bar of *danskebank.dk*, which shows the *"become a customer"* link.

The task is of interest to Danskebank, as it can help identify visitors, who exhibit the behavioural traits of someone wanting to sign up. Knowing this allows the bank to proactively reach out to the visitor and show special offers or a pop-up with a link to the sign up options. We believe that the following features in the Celebrus data will help us solve the task:

- Visitors, who previously clicked on the link, will most likely have a similar click path - perhaps the logical flow of the webpage directs visitors in a specific click path before clicking *"become a customer"*. Thus we believe that the click path of the visitors will help us solve the task. Table 7 shows the most visited URL's for visitors who clicked and did not click on the "*become a customer*" link. The table was constructed, by taking the set difference between the top 40 most frequent URL's for each group and then showing the top URL's on this list.

Table 7: Most frequent links for visitors who clicked and did not click on the "*become a customer*" link. the prefix *danskebank.dk/privat* has been stripped from each URL.

Did not click	Clicked
/produkter/laan/bolig-laan/andelsbolig-laan	/find-hjaelp/konti/oprettelse-af-ny-konto
/produkter/laan/bolig-laan/obligationslaan	/produkter/konti/konto-for-unge-og-studerende
/find-hjaelp/valuta/valutakurser	/find-hjaelp/ris-ros-og-klager#book-a-meeting-skriv-form

  From the list of most frequent URL's, we see that the visitors who did not click the link searches for information about different loans and currency prices, whereas visitors who do click on the link searches for information about creating a new account and booking a meeting.

- We also believe that the time between clicks will help us solve the task as visitors, interested in becoming a customer, might search information regarding different products of the webpage, resulting in long time between clicks, due to unfamiliarity with the page and reading of information.

For the task, we will only consider visitors who are not logged in and does not login during their session, meaning we will only consider a small subset of the entire data. Only a fraction of these visitors click the

"become a customer" link and the class is therefore heavily under-represented. The frequencies of the two classes in the data is shown in table 8

Table 8: Frequencies of the two age groups in the dataset.

Clicked	**Did not click**
0.0293	0.9706

7 Processing Pipeline

In order to get from the raw Celebrus data, comprising several tables, and being of varying length and type, to data with uniform dimension, that can be fed to a model, the data goes through several stages of refinement. Practically, the relevant data is featurized by applying transformations to the Celebrus data, and assigning each sessionnumber a feature vector of uniform length. We have decided to modularize these steps, and have built our workflow using a pipeline as the fundamental abstraction for handling the data.

Since the amount of data, to be processed, is too great to fit into the memory of our edge node, we use Spark to process it. Using Spark is also convenient, because the data resides in a Hive database on Hadoop, which Spark can address. When pointing to an external database, Spark can encapsulate a table in a DataFrame. This construct comes with the convenience of being able to execute SQL-like statements expressed using a fluent API, i.e. making expressions of the form:

Listing 1:

```
1 danish_page_clicks = page_table.
2         select(page_table.
3                 pagelocation).
4         where(page_table.
5                 pagelocation.
6                 like("%danskebank.dk%"))
```

This makes it possible to iteratively build SQL statements, and also has the advantage of expressing computations declaratively - i.e., by not specifying the control flow, but the logic - and thereby having the query optimizer worry about handling computational details, and optimization.

Because Spark lacks some machine learning functionality, otherwise available in other Python libraries, and time does not allow the development in this framework, the pipeline is split into a Spark part, and a Python part. An overview of the pipeline can be seen in figure 12. The steps in the pipeline will be discussed in details below.

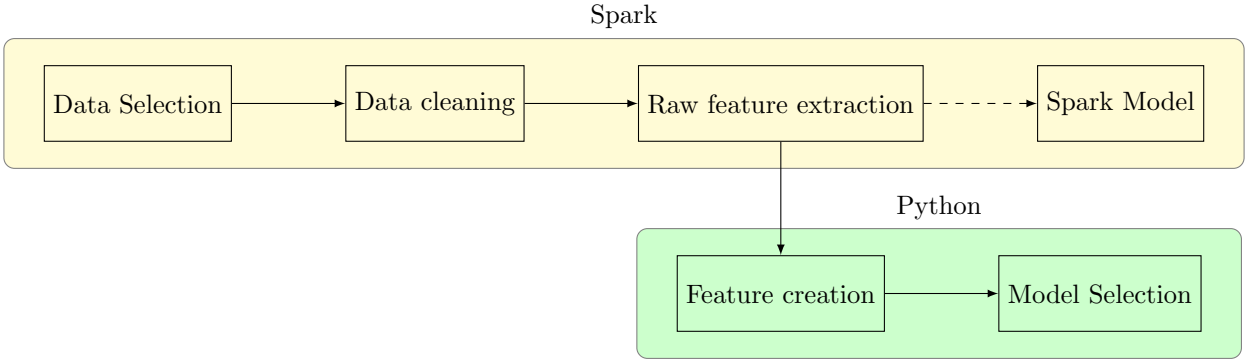

Figure 12: Illustration of our processing pipeline.

7.1 Data selection

The overall purpose of this step is to filter out unwanted data, such that transformations are applied only on relevant data.

The targets for the first and second task comes from an analytics (ABT) table. This table contains aggregated information about customers. In the ABT tables and almost all other systems in the bank, a column named `knid` is the primary key. This is a unique 10 digit number, assigned to each customer. Celebrus contains a column called `profileuiid`, which within Danske Bank is know as the ebanking agreement number. Every customer has a single ebanking agreement number, which might change in time.

A mapping from `knid` to `profileuiid` has to be established. This mapping exists in a table called `gard_ewwh_ebank_agree_h`. The table is continuously updated every month, and since the `profileuiid` might be changed (due to the customer changing hers agreement with the bank), it is important to note, that the mapping between an `profileuiid` and a `knid` (ultimately a customer), is a many-to-one - even if the customer only has a single active `profileuiid`, the previous `profileuiid` of the customer might still have data related to it in Celebrus.

The rows in `gard_ewwh_ebank_agree_h`, and ABT are not modified. At the end of each each month, rows are added to the table. Therefore we are interested in the data, that lies as close in time as possible, to the Celebrus data. Thus, a customer browsing the website in December 2017, might have had birthday since then, and that session, should be associated with the age of the customer at that point in time.

Once there exists a mapping between the ABT table and an `profileuiid`, we take the intersection of this `profileuiid` with the Celebrus data. The data selection part of the pipeline returns a dataframe, with columns:

$$\texttt{sessionnumber}, \texttt{profileuiid}, \texttt{knid}, \texttt{target}.$$

We now have a complete mapping of `profileuiid` to `knid`, together with a target that might differ over the same `profileuiid` according to the time of the session.

7.2 Data cleaning

This step is concerned with applying simple transformations to data, joined with the output of previous pipe.

The output from the data selection pipe, is joined on the relevant tables. The full URL's that appears in the browser as the user navigates the website, is available. The problem with these URL's are that they contain information that are unique relative to the individual session. An example of some URL's can be seen below:

$$\begin{aligned}
&\text{https://netbank2.danskebank.dk/REB/HFAnsoegDI/FrontPage.aspx?q=7549993...}\\
&\text{https://netbank2.danskebank.dk/REB/HFAnsoegDI/FrontPage.aspx?q=6811233...}\\
&\text{https://netbank2.danskebank.dk/REB/HFAnsoegDI/FrontPage.aspx?q=1010984...}\\
&\text{https://netbank2.danskebank.dk/REB/HFAnsoegDI/FrontPage.aspx?q=3010985...}
\end{aligned} \tag{1}$$

Note that the "..." means that the URL continues. There exists exactly one occurrence of each of the above URL in the Celebrus dataset. The URL's contains parameters which are unique and not meaningful in regards to specifying the users click path.

In the raw Celebrus data, we have found over 22 million unique URL's, and the vast majority of these URL's, are only used once. What we are really interested in, is tracking a user across different pages on the Danske Bank website: If two or more different visitors clicks on a specific page, they should be registered, as being on the same page, as opposed to having a unique identifier, as is the case in (1). It can be seen in (1) that the URL's differs only after "q=". If this information is removed, then the URL's can be used to specify where on the page a user is located. Some regular expression rules have been applied to remove unwanted information from the URL's, these are listed in the following table:

RegEx	Removes
\s+	Any whitespace character
\? (.*)	All after and including question mark
[# | &]$	Any trailing # or &
^(https | http | www.)	https, http or www. if in start of URL
\/\/	Double forwardslash, replaces with single
[\/|,|.|'|-]$	Removes any training characters of "/", ".", "", "-"

Table 9: All the regular expressions on the left, are substituted with the empty string.

Applying the regular expressions from table 9 drastically decreases the number of unique URL's. The number of unique pages, after applying the regular expressions was roughly 5000, as is shown in figure 13, which shows the number of clicks made on each unique page, as a function of the pages sorted by frequency in descending order. The number of sites which has more than a few clicks, is relatively low.

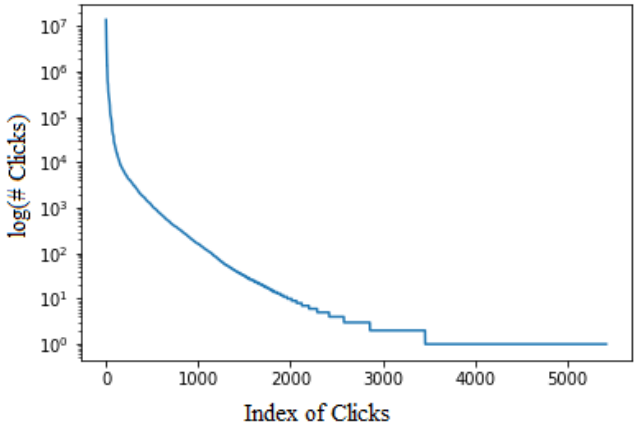

Figure 13: Log plot of number of URL's after regex transformation, x axis means index for each URL, y axis is the times for each URL appears

The reason for the many unique URL's remaining, is that the regular expressions only removes text that matches a predicate. There exists thousands of examples of users who have typed in text in the URL themselves, which is still registered in the Celebrus data. The way these URL's are removed, is to make a hard restriction on the number of times a URL (with regular expression removal) should appear, before it is considered usable - in this project, a page needs to have been visited at least 3000 times, otherwise removed for further processing. The final cardinality of the number of unique pages is 336. After the normalization and removal of unwanted URL's, adjacent pages, i.e. identical pages occurring multiple times in a row are removed, thus only one of these pages is kept. Lastly, sessions which contains less than 6 clicks are removed.

A website is dynamic and content may be added or changed, resulting in new URL's appearing in some time frame, as well as URL's disappearing. Thus, the data reflects how the website looked, at that particular point in time of the click, and does not take into consideration, that the page might have changed. Because we only have data from a period of 4 months, and the website is rather static and not subject to major changes, no actions have been taken to remedy this. Specifically, we are considering data from 01-01-2018, to 01-05-2018.

The output of the data cleaning pipe is a dataframe, with the following columns:

`sessionnumber`, `pagelocations`, `timestamps`, `target`

Where `pagelocations` and `timestamps` are lists of variable length, holding the list of visited URL's and timestamps for the visits respectively. A Spark DataFrame can facilitate processing of data that has varying length, by encapsulating a list in single field in the DataFrame.

7.3 Raw feature extraction

When the data is cleaned, there now exists a mapping from sessionnumber to a list of URL's and timestamps. The final step of the Spark part of the pipeline is twofold: First, gathering features which needs not be cleaned further, and second, dump the result to a file, in CSV format.

The gathered features are as follows:

- Time of visit, which is stored under three columns as `day`, `hour` `minute`
- Operating system (os)
- Browser

Thus the output from the feature extraction pipe is a Dataframe, with the following columns.

`sessionnumber`, `pagelocations`, `timestamps`, `day`, `hour`, `minute`, `os`, `browser`, `target`

The Dataframe, holding the above mapping from a sessionnumber to a set of features for all sessions, is randomly subsampled and written to disk on the edge node, in the form of a CSV file.

7.4 Feature Creation

The former steps has extracted and transformed various columns, from different tables of the Celebrus relation. For this step of the pipeline, an ETL (Extract Transform Load) library has been developed, in order to get the features to their final form.

This pipe comprises three logical steps:

1. Extract the data from CSV into memory.
2. Transform the data.
3. Load the data to .npy format where all session mappings has uniform length. This format is readable by the Python machine learning frameworks we use.

This second iteration of processing was necessary in order to apply transformations to the data, which are not easily expressible in Spark. Working in Python also allows us to prototype rapidly, i.e. try different flavours of features - For example, should time be z-score normalized according to click, or according to page category? Is the click path best represented as a one-hot encoded vector, or Bag of Words?

7.5 Model selection

When the data has been cleaned and featurized, machine learning models can be applied. In this project, two tracks of machine learning models are represented - traditional ML methods such as Random Forest on the one hand, and a deep learning model on the other.

8 Features

In this section we will describe the features that were extracted from our Spark pipeline and go through how they are represented in our models.

8.1 URL

Perhaps the most important component, when modelling visitor behaviour through the Celebrus data, is the click path of a visitor. We define a click path as the sequence of webpages visited by the visitor, listed in the order of visit. This sequence will vary in size for visitors, depending on the number of clicks made on the webpage. A webpage is represented as a URL or *Uniform Resource Locator*, which is used to specify the location of a file on a network of computers, most commonly used to specify the location of a webpage. By looking at the URL we can uniquely identify what webpages the visitor visits. Dependent on what behaviour is being modelled, click-paths can give away certain tendencies of a visitor. We believe there are two attributes of the click-path to consider when representing them:

1. **Content of click**. The content on the webpage, i.e. what kind of information is stored on webpage, specified by the URL. This can for example be used to determine what interests the visitor has, and what information the visitor is looking for, but also to group visitors together, based on what content they viewed.

2. **Context of click**. Taking into account the entire sequence of clicks, thus considering a click in the context of the other clicks in the click path. This can be used to evaluate the intuitive flow of the webpage but to detect behavioural traits such as fraudulent behaviour.

Thus we wish to obtain a representation of the clicks, which takes into account the content and context of the clicks as we believe including these will give better results. To verify this, we will compare the results when using different representations of the clicks, in order to verify the above hypothesis. From our Spark pipeline, we obtain an ordered list of preprocessed URL's for each session, which represents the click paths of each session. We then wish to obtain some alternative representation of these click paths that can be used by our machine learning models. The representations that we have obtained are as followed:

1. Bag of Words.

2. Word2Vec.

3. Learning features by means of a neural network.

Below we will go through the Bag of Words and Word2Vec representations. We will describe the last representation in section 9.2.1.

8.1.1 Bag of Words

The Bag of Words representation (BOW) is a simple representation method, commonly used in natural language processing as a way of representing the presence of words in a document, alongside their frequencies. The representation uses a vector with length equal to the size of the entire vocabulary, where each entry in the vector refers to the count of the corresponding word in the document. For example take the following sequence:

$$[A, A, B, A, C, D, E, B]$$

Given the vocabulary:

$$[A, B, C, D, E, F]$$

yields the following Bag of Words representation:

$$[3, 2, 1, 1, 1, 0]$$

The drawbacks of the approach is that the order of which the elements appeared in the sequence is lost and therefore does not take into account the context of the element. Another drawback of the method is that vocabularies tend to grow large, which results in a sparse representation and can be computational expensive to process.

To use Bag of Words on a single session, we create a one-to-one mapping of a URL to an integer, in the range [1;336], as there are 337 unique URL's. We then create a vector of size 337 for each session, where each entrance is a count of the occurrence of an index, like in the example above.

8.1.2 Word2Vec

Word2Vec is a group of models for word embedding which consists of a three-layer neural network; it contains an input layer, a hidden layer and an output layer[2]. The skip-gram variant of word2vec takes as input a word and predicts its context. The training goal of this model is to learn a word vector representations that have the highest probability of predicting neighbouring words. In this project, the skip-gram model is used to represent features.

The skip-gram model has two input parameters: Window size and embedding dimension. Suppose we have a document, and split each sentence in the document. In the training process, the model learns to predict its neighbouring words according to the window size. e.g, if the window size is a positive integer n, then for each word, the model learns to predict the n words before and after a particular word. Before training starts, the input vector is one-hot encoded. The length of the vector is dependent upon the size of the corpus, that is, for some $x_i = 1$ and all the other $x_{i'} = 0$ for $i \neq i'$. e.g. if the corpus is "animals", "dog", "cat" and "fish", which contains four words, the one-hot encoding for each word is [1,0,0,0] for "animals, [0,1,0,0] for "dog", [0,0,1,0] for "cat and [0,0,0,1] for "fish".

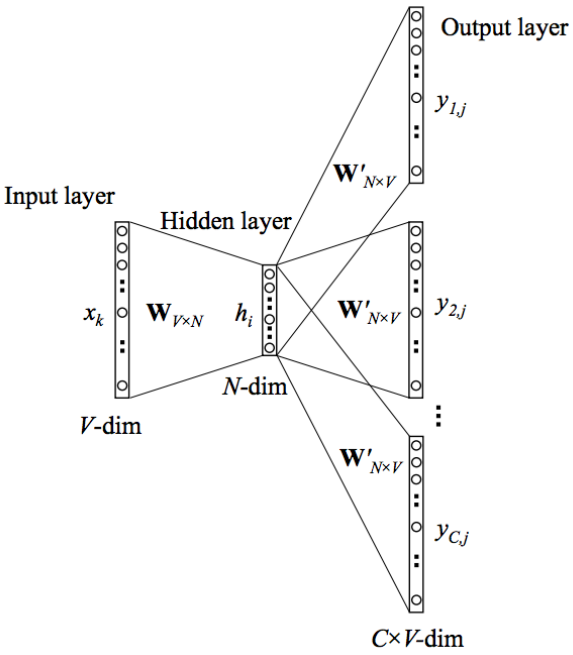

Figure 14: Visualization of Word2vec. Taken from `http://wuciawe.github.io/machine%20learning/math/2017/02/12/notes-on-word2vec.html`

For the network of this model, assume V is the size of this corpus (i.e. there are V unique words in all documents) and N is the size of the hidden layer (embedding dimension), as can be seen in figure 14. The input vector $\mathbf{x} = [x_1, x_2, \cdots, x_V]$ is one-hot encoded for $x_i = 1$. The weights between the input layer and hidden layer is a $V \times N$ matrix. Every row in $\mathbf{W}$ is denoted as $\mathbf{v}_{w_i}$ and is an N dimensional vector representation.

$$\mathbf{h} = \mathbf{x}^T\mathbf{W} = \mathbf{W}_{(i,:)} := \mathbf{v}_{w_i} = [w_{i,1}, w_{i,2}, \cdots, w_{i,N}]. \tag{2}$$

Since $\mathbf{x}$ is one-hot encoded, $\mathbf{h}$ is the i'th row of weight matrix $\mathbf{W}$. This implies that the activation function between the input layer and hidden layer is linear, and $\mathbf{W}$ can be seen as a lookup table. From the hidden layer to the ouput layer, there is a different weight matrix $\mathbf{W}'$ which is a $N \times V$ matrix and we can use this matrix to calculate the score for each word in corpus. We define $u_i := \langle \mathbf{h}, \mathbf{v}'_{w_i} \rangle$ where $\mathbf{v}'_{w_i}$ is the i'th column of $\mathbf{W}'$, and u_i is the score of i'th word for measuring the match between it and its neighbour word within our defined window size. Then a softmax classifier is applied, which is log linear to get the probability distribution of each word, as shown in (3):

$$p(\text{word}_j|\text{word}_i) = \frac{e^{u_j}}{\sum_{k'=1}^{V} e^{u_{k'}}} \tag{3}$$

After the training precedure is done, $\mathbf{W}'$ can be discarded.

Word2vec is in our project used by creating a one-to-one mapping from a URL to an integer in the range [1;336] - like the Bag of Words model. Each click path of a session is then represented by a list (of varying size) of indices, which maintains the order of the clicks. After this step, Word2vec can be directly applied.

8.2 Url time

We log the timestamp for each click, made by the visitor. The timestamp identifies the date time for when the click was made. The processing pipeline outputs a sequence of unix timestamps, which is processed by

taking the difference between adjacent timestamps, resulting in a sequence of seconds between each click. This sequence is then represented as followed.

- **click_time**: A cut-off is made in the time to form a 10×1 dimensional vector where the i'th entry represents the time spent on the i'th click in the session. For sessions with less than 10 clicks, the vector is padded with zeros. The z-score (zero mean and unit variance) is taken for each column of the feature matrix, resulting in the features.
- **avg_clicktime** The z-score of the average time between clicks is taken for each visitor.
- **std_clicktime** The z-score of the standard deviation of the time between clicks is taken for each visitor.

8.3 Time of visit

From our pipeline the time of visit is extracted. The time of visit consists of two measures:

- **Time of day**: which is represented in minutes, in the interval [0, 1339]
- **Time of week**: which is represented in days, in the interval [0,6]

We want to represent time as a clock, for example we want minute 1339 and 0 to have the same distance as minute 0 and 1. To obtain such a representation, we used sine and cosine functions to represent our interval on a unit circle. This representation is calculated as followed:

$$[\sin(2\pi \text{ time } / \max(\text{time})), cos(2\pi \text{ time } / \max(\text{time}))] \tag{4}$$

where time is either the minute of the day or the day of the week. $max(time)$ is the largest value that the respective time can take. The representation of the two time measures are each a two-dimensional feature vector where both entries takes values between 1 and -1. Thus the representation is a 1×4 feature vector.

8.4 OS and browser

We retrieved the operating system and browser of each session. To reduce the number of operating systems, we grouped together different versions of the same operating system. For example, all Linux distributions were grouped together, all Windows operating systems before and including Windows 7 were grouped, likewise Windows 8, 10 were grouped together and so on. Operating systems who appeared in the data less than 100 times were dubbed as category "Other". For browsers, the same approach was applied. For this approach, we ended up with 7 types of operating systems and 7 types of browsers. The categories were as followed:

Operating system	Windows old	Windows new	Linux	MacOS	iOS	Android	Other

Table 10: List of obtained operating systems.

Browser	Edge	Internet Explore	Firefox	Safari	Chrome	Opera	Other

Table 11: List of obtained browsers.

Both operating system and browser are represented as an index into a one-hot encoding of the specified browser index, thus both features takes an integer in the range [0,6].

9 Models

In this section we will describe the models which we used to solve the selected tasks and why we chose these models. We wanted to try out two different approaches:

- Using an Established and well documented machine learning method.
- Using a deep learning model, which includes an alternative representation of our features. This model is obtained on a trial-and-error basis, inspired by various approaches in the field of deep learning.

We tried a variety of machine learning models, to identify the one which performed best on the given tasks. We found that of all the models, the Random Forest method performed well for each of the tasks, therefore we chose this model. We also chose the model as it is intuitive and easy to inspect what the model is learning. We do not know which of the generated features can be used to solve the tasks if any, Random Forest is well suited for this situation as it performs features selection on it's own. Random Forest also allows us to easily see what features are most important, for the given task, which is helpful as we have never worked with the data before.

9.1 Random Forest

Random Forest is an ensemble method, which means that the model uses multiple machine learning models, also referred to as "weaker learners", to output a single hypothesis. This final hypothesis is decided based on the predictions of the weaker learners e.g. by performing a majority vote for classification, or taking the average predicted value for regression. Random Forest constructs a number of decision trees and uses the predictions of these trees to output a final prediction. In this section we will briefly go through how decision trees work for classification and regression, and how random forest works.

9.1.1 Decision trees

Decision trees builds a binary-tree like structure by repeatedly dividing the data into smaller datasets, based on some decision on a feature. This division is repeated until the dataset is small enough or all labels in the reduced dataset belong to the same class / are similar enough. When this criteria is satisfied, the splitting stops and the node becomes a leaf, which defines a decision region in the input data. More formally, after a decision tree is created, each internal node is associated with a feature index $i \in \{1, \dots d\}$ and a threshold θ, which combined defines the decision of the internal node, i.e. whether to visit the left or right subtree. This decision is defined as followed for the training data S:

$$L_{i,\theta} = \{(\mathbf{x}, y) \in S | x_i \leq \theta\} \text{ and } R_{i,\theta} = \{(\mathbf{x}, y) \in S | x_i > \theta\} \tag{5}$$

As we split the data, we wish to produce decisions such that the uncertainty of the prediction at a node is reduced. This uncertainty can be described by an impurity measure, which quantifies how mixed the data is. The information gain uses the impurity measure, to identify how much the uncertainty is reduced, or how much the dataset is purified, given a split on feature i with threshold θ. The information gain is defined as followed:

$$G_{i,\theta} = Q(S) - \frac{|L_{i,\theta}|}{|S|} Q(L_{i,\theta}) - \frac{|R_{i,\theta}|}{|S|} Q(R_{i,\theta}) \tag{6}$$

where Q is the impurity measure. As evident from (6), the uncertainty is weighted after split size, thus we care more about the impurity measure for larger splits. To achieve an optimal decision tree, we wish to identify the splitting feature i and threshold θ, at each node, such that the information gain is maximized. To find this optimal split, we calculate the information gain at each feature split, and chose the one which maximises the information gain. When dealing with continuous variables, the practice is usually to sort the

data points on the given feature and discretize the continuous features into intervals. The information gain is then calculated at each interval. The impurity measure used differs, dependent on the given problem. For our classification tasks, we have chosen the gini-impurity. The gini-impurity is defined as followed:

$$Q(S) = \sum_{c=1}^{C} p_{Sc}(1 - p_{Sc}) \tag{7}$$

Where p_{Sc} is the fraction of samples in S which belong to class c. The gini-impurity quantifies the likelihood of performing an incorrect classification when drawing an observation from the dataset, when this observation is classified according to the distribution of the classes in the set. Thus if our dataset only contains elements of class "A", then the probability of performing an incorrect classification is zero as all the drawn and predicted elements are class "A". If, however the dataset is perfectly balanced, the probability of misclassifying this element is 0.5. For regression, we are dealing with continuous output variables, and therefore the impurity measure must support this. The impurity measure used for the regression task is the mean squared error, defined as followed:

$$Q(S) = \sum_{(x,y)\in S} (y - \hat{y})^2 \tag{8}$$

where

$$\hat{y} = \frac{1}{|S|} \sum_{(x,y)\in S} y \tag{9}$$

Thus reducing the above impurity, reduces the variance of S, with respect to the target value. Once the decision tree is built, new instances are predicted by simply progressing down the tree, adhering with the rules at each node, until a leaf node is reached, which gives the predicted output. For regression tasks, the usual practice is to take the mean of all the training labels in the leaf.

9.1.2 Random Forest

The Random Forest model takes as input a number of decision trees to build: T. The method first creates T bootstrap samples, i.e. T randomly sampled subsets with replacement, each with same size as the training data. For each of the bootstrap samples, a decision tree is built. When splitting a node, during construction of the decision trees, the model only considers a random subset of the features to split on. To perform predictions, the model uses bagging i.e. uses an aggregation of the output of the different classifiers to output a single decision. This decision was a majority vote amongst the decision trees for the classification tasks and an average of the outputs for the regression task. Aggregating the outputs of the randomly build decision trees, helps reduce the variance of the model i.e. prevents overfitting. For the tasks we used Sklearn's[3] RandomForestClassifer[3] and RandomForestRegressor[4] implementation. For each model, we built a forest of 100 decision trees, which was chosen as it gave a nice trade-off between performance and runtime and increasing this number further did not approve the performance much. Tuning the parameters of the model was tried, more specifically we tried forcing constraints on the max-depth of the trees, the minimum samples per split, the minimum number of samples required to form a leaf and constraints on the required decrease in impurity of each split. We found that setting these parameters only worsen the results and eventually we chose to use the default parameters of the model, which does not force the above constraints. We also use the models to estimate importances of the features. Sklearn estimates the importance of a feature i by summing up the information gain achieved for all nodes, when splitting on feature i, divided by the total achieved information gain across all nodes[5].

[3] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[4] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
[5] https://github.com/scikit-learn/scikit-learn/blob/0abd95f742efea826df82458458fcbc0f9dafcb2/sklearn/ensemble/forest.py

9.2 Deep model

A deep model has been created, with the purpose of competing against traditional machine learning methods. The general idea of the model, is to first learn a representation of each feature component, described in the Features section, and then concatenate these to create a combined feature vector. The feature vector can be given as input to any model. A full view of the model can be seen in figure 24 in the appendix, together with a technical description of the model generated in Keras[4].

The deep learning architecture is created in Keras using TensorFlow as backend. Keras is used, because it is relatively easy to prototype in, and test different architectures: It provides a high level API, where every layer in a network is treated as an object that is easily manipulated, and thus abstracts away error prone matrix multiplications. The `layers` modules provides several components of a deep learning architecture, and in this report, all the components of the described network originates from said module. Below we will describe how the features were represented for this model.
The model is run on the GPU of the Tesla server, described in the technical frameworks section.

9.2.1 Representation of clickstream data

There are several ways, the click path of a given user can be represented. This project has applied a sequence processing/NLP approach to the problem, and is inspired by Yoon Kim, as described in [1], which performs sentiment analysis on various benchmark datasets, and compares them with other recent methods. The model proposed in [1], outperforms the other models on a subset of the benchmarks. The proposed model consists of a word embedding of an input sentence of size n, zero padded - ie. if a sentence is shorter than n, 0 is appended, such that every sentence is of uniform length. The embedding is convolved with 1-dimensional filters, and then max pooled. The result is a feature vector, which is used in a fully connected network.

By observing the legal transitions between webpages that can made on Danskebank.dk, it can be seen that in some way, this graph resembles a grammar: There exists a limited number of possible transitions at each point. This means, that when navigating the webpage, it might be possible to go from page A to B to C, but not from page A to page C. Therefore, it was decided to treat each URL in a click sequence, as a word in a document. Each URL has been encoded with a number in the range [1;336] (the number of unique URLs), and maximum 40 visited URLs are considered, zero padded where necessary. If a session has more than 40 clicks, it is downsized to 40.

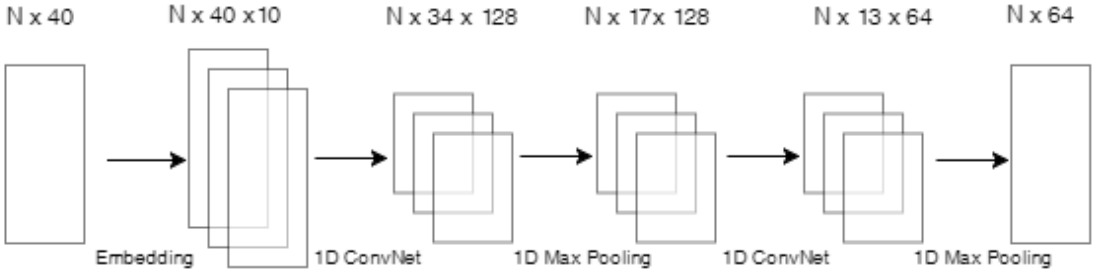

Figure 15: Representation of the click path.

Figure 15 shows the architecture for click representation. Initially the input consists of N batches, which is one-hot indexed (ie. every entrance in a vector is the index of 1, in a one-hot encoding). The first layer is an embedding layer, which works similar to the Word2vec model described. It acts as a lookup

table, for each click encoding, assigning every entrance in the input, a vector of length 10. The weights in the embedding layer are initialized uniformly at random, and learned through backpropagation. Whereas skip-gram Word2vec learns a word representation given contexts of words and is independent of the task, this embedding learns a representation subject to a specific task. Through experiments it was found that representing each click as a 10 dimensional vector gave best results.

1-dimensional convolution was then applied to the click embeddings. Borrowing notation from [1], for each click path representation, ie. every 40x10 matrix, 1-D convolution can be described mathematically as:

$$f(\mathbf{w}^{\mathrm{T}} \cdot \mathbf{x}_{i:i+h-1} + b) \tag{10}$$

Where each $\mathbf{x}_i \in \mathbb{R}^k$. $\mathbf{w} \in \mathbb{R}^{kh}$. b a bias. Colon is the concatenation operator, and f is a non-linear activation function. $\mathbf{x}$ corresponds to a single row, and h denotes the number of consecutive words to consider - also called the window size. In this report, a window size of 3 was chosen. This practically means, that every three subsequent rows are made into a vector by placing them side by side, and then dotted with a vector, $\mathbf{w}$, of the same dimension. 128 filters are used for each layer - figure 15 displays a single convolutional layer, but 3 was used, all with 128 filters. Padding has not been used, therefore for each convolutional layer, 2 dimensions are lost.
Convolution, especially 1-dimensional, is a cheap operation, requiring only $kh + 1$ parameters per filter. Instead of convolution, an LSTM, GRU, or other recurrent layers could have been used - these have been used successfully in NLP applications, yielding good results [5] . These were tested, both as a replacement for convolution, and as an extension, but did not yield satisfying results. The following layer is a 1-D max pooling layer, of size 2, and stride 2. It computes the max value, along the second dimension. Then, a dropout layer is used for regularization, dropping 20% of the values. Following the dropout is yet another 1-D convolutional layer, with 3 x 64 filters, followed by a global max pooling. Global max pooling reduces the rank of the tensor by 1, by computing the max of the second dimension. A click path is thus represented as a vector of length 64.

9.2.2 Representation of time between clicks

The Z-score of the time between clicks is used as input. This time encoding experimentally gave the best results.

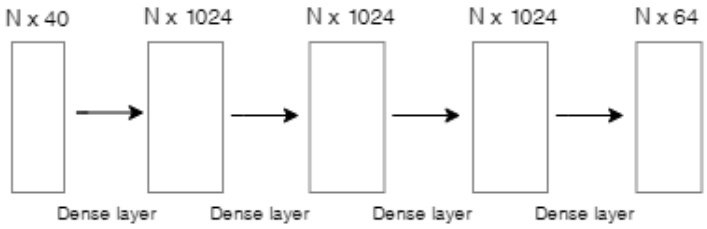

Figure 16: Representation of time between clicks.

Figure 16 shows how click time is represented. A maximum of 40 clicks are considered and given as input to a fully connected network, with 1024 weights, followed by two layers of 1024 weights. The output is a 64 vector, which is the feature vector representing time between clicks, for each click path. Between each of the networks dropout with rate 0.5 is used as regularization.

9.2.3 Representation of Browser and OS

The same principle applies for the representation of OS/Browser, as does for a click path. OS and browser are both one-hot indexed. OS $\in [1; m]$, and browser $\in [m+1; l]$ for a single session. m denotes the number of

distinct OS', and $l-m+1$ denotes is the number of distinct browsers - the intersection of the two encodings is empty, allowing the use of an embedding layer without clashes between joint embeddings of OS and browser. Since OS and browser are somewhat related, they have here been combined to create a single feature. OS is a strong indicator for what browser is used, and vice versa.

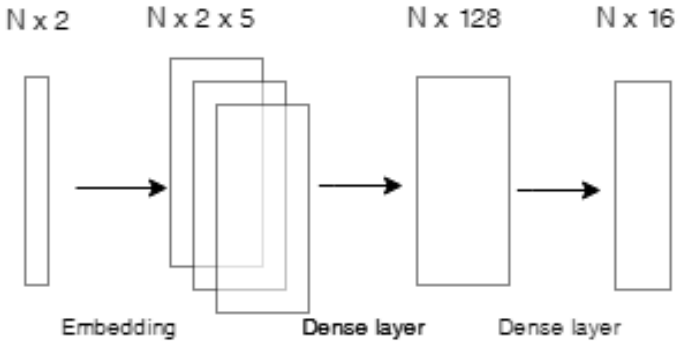

Figure 17: Representation of OS and browser.

Figure 17 shows how OS and browser are combined in a joint representation. The first layer is an embedding layer, outputting for each session, a 2x5 matrix, making a dense representation of the OS and browser respectively. The following two layers are fully connected, employing dropout with rate 0.2 - the low drop rate was chosen, due to the small amount of weights, relative to the two other models. The output for each session, is a vector of length 16.

9.2.4 Combining the features

Besides the features explained above, Login time of day and week, mean time between clicks, and click time standard deviation, in total 6 extra values, are also used as features. This yields, for each session, a $64 + 64 + 16 + 6 = 150$ feature.

The features are combined by a fully connected layer with dropout, as can be seen in 18.

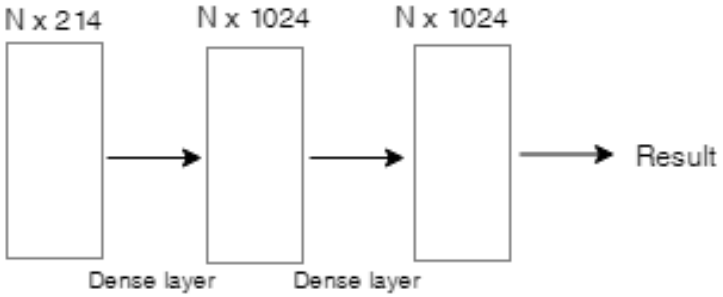

Figure 18: Last layers in the deep learning model.

The value of the output "Result" depends on the task. For the regression task, the output is a linear combination of the outputs of the previous layer, and for binary classification (the two other tasks) the output is the sigmoid function, of a linear combination of the previous layer, defined as such:

$$\text{linear}(x) = x \tag{11}$$

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{12}$$

Note that for all layers, ReLU has been chosen as the activation function:

$$\text{ReLU} = \begin{cases} 0 & \text{if} \quad x < 0 \\ x & \text{else} \end{cases} \tag{13}$$

Also, all the layers for the different feature representations, are learned by means of backpropagation, with respect to the target.

The space of hyper parameters for the network presented is huge. Therefore, it is not feasible to perform hyper parameter optimization in regards to the number of weights in a given layer. The decisions regarding the representation of the features was taken on a naive basis: For each feature, different architectures was tried disregarding all other features - the feature representation that gave the best result, was chosen.

10 Experimental Setup

In this section we wish to describe the setup of our experiments. Here we will outline the combinations of features and models used, how the results were evaluated and how they were compared.

We have a fixed set of features, as described in section 8, which consists of the following:

- **Click-path:** Here we wish to try out three different representations of the click-paths. The representations are the following:
  - **Bag of Words:** 1×336 Bag of Words representation.
  - **Deep Features:** 1×64 feature vector, learned by the deep learning model.
  - **Word2Vec:** 1×10 Word2Vec representation, using a window size of 3, and embedding size 10.
- **Click-path-time:** 1×10 feature vector, representing the normalized click-time of the first 10 clicks.
- **Time of visit:** 1×4 feature vector, representing the time of day and time of week of visit.
- **Click time:** 1×2 feature vector. The first entry represents the average time between clicks. The second entry represents the standard deviation of the time between clicks.
- **OS-and-browser:** 1×2 feature vector, representing the operating system and browser used.

Thus the feature vector is of size $|\text{click representation}| + 18$. We have two models, as described in section 9, which we will use for prediction:

- **Random Forest**: Here we will use the above described features. We will apply the model with the three different representations of the click-path.
- **Deep Learning model**: Here we will use the learned representation of the features, obtained through the deep model, as described in section. We will apply the model with the three different representations of the click-path.

Thus in total we will perform six experiments for each task.

10.1 Evaluation methods

Here we will describe how the experiments were evaluated.

10.1.1 Regression

In this section we will describe how our regression problem, i.e. task 1 is evaluated. Given a set of labels y and a set of corresponding predictions $\hat{y}$, the accuracy is evaluated as followed:

$$\text{acc} = \frac{1}{|y|} \sum_{i=1}^{|y|} M(y_i, \hat{y_i}) \tag{14}$$

where $M(\cdot)$ is defined as followed:

$$M(y, \hat{y}) = \begin{cases} 1, & \text{if } |\hat{y} - y| \leq 5 \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

Thus a correct prediction is made when the prediction is within 5 years of the true label. This interval was chosen rather ad-hoc, but was deemed acceptable as this seemed like a natural cut-off, for when behaviour starts differing.

10.1.2 Classification

For each of the classification tasks, we are interested in determining the performance of our model on the single class we are trying to predict, that is for task 2 we wish to correctly classify visitors in the age range [18-27] and for task 3 we would like to predict whether a customer will click on the "*Become a customer*" link. Thus we will not use a simple binary accuracy measurement, which measures the overall performance on both classes, but rather one which identifies the performance of the model on the single class. For such a measurement we will use a precision-recall curve and base the overall performance of the model by the area under this curve. Precision is defined as followed:

$$\text{precision} = \frac{TP}{TP + FP} \tag{16}$$

Where TP is True Positive: The number of correct predictions of the positive class, and FP is False Positive: The number of wrong predictions, of the positive class. Thus precision is the number of times the positive class was predicted correctly, divided by the number of times it was predicted. Recall is defined as followed:

$$\text{recall} = \frac{TP}{TP + FN} \tag{17}$$

Where FN is False Negative: The number og wrong predictions, of the negative class. Recall is the number of times the positive class was predicted correctly, divided by the number of instances belonging to the positive class. Whereas precision can increase and decrease, recall is a monotonically increasing function as the number of positive observations for the positive class is constant, as a function of the number of data points considered. The precision-recall curve shows the relationship between precision and recall for different probability thresholds. The idea behind the Precision-Recall measure, is to make the threshold variable. For every threshold value, all classifier prediction probabilities above the threshold is classfied as positive, and all lower are classified as negative. For each threshold value, precision and recall is calculated. To produce a single point measure of the methods, we have decided to use the area under the precision-recall-curve as this gives us an overall performance from the model.

10.2 Baseline

To compare the models, we wish to compare the results against some simplistic predictions. Below is a description of what baseline method will be compared against, for each task

- **Task 1**: At each prediction, the baseline predictor predicts the most frequent age in the training data.
- **Task 2, 3**: At each prediction, the baseline predictor outputs a random probability between [0,1]

Comparing against the above baselines will reveal if the given model is actually learning something from the data or just outputting random predictions.

11 Results

This section will present the results of task 1, 2 and 3. For the tasks, we use the features presented in the Features section, as model inputs. For each task, a table is presented which summarizes the results, according to the evaluation measure, presented in the previous section. The leftmost column of the tables refers to the representation of the click path.

For each task, a table summarizing how the model was trained is presented. The Train column shows the number of data points used to train the models, and the Test column shows how many data points were used to evaluate the models. When the Random Forest model trains/tests on the Deep Features, it uses the same click representation, as the Deep Learning model. The Sampling Method column tells which technique was used to sample the data points used: Balanced means a stratified sampling technique was used, where each strata (age in the interval $[18; 84]$), contains the same number of samples. Unbalanced means that a random sample was taken from the dataset.

11.1 Task 1

For task 1, we wish to predict the age of a customer browsing danskebank.dk.

	Training	Test	Validation	Sampling method
Deep Learning Model	965118	451431	501590	Balanced
Random Forest	834372	147243	-	Unbalanced

Table 12: Task 1, number of data points used, and sampling method.

Table 12 shows the number of data points used, together with the sampling method.

	RandomForest	Deep Learning	Baseline
BOW	0.2934	0.309	0.2714
Deep Features	0.29371	0.313	0.2714
Word2vec	0.2954	0.2906	0.2714

Table 13: Task 1 results

The results from task 1 are summarized in table 13.

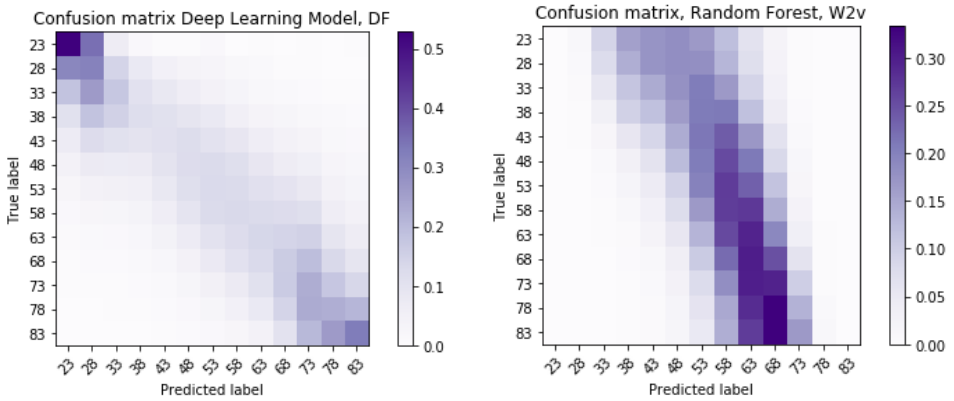

Figure 19: Confusion matrix show results for the deep learning model using the deep learning features on the left, and the random forest model using Word2vec on the right.

The best performance was the deep learning model, using the Deep Features. For both models, there is a small difference between results, across the different features. The biggest difference is seen between the deep model using Word2vec and the Deep Features. The difference between the features for the random forest model is differing only by a small fraction.
According to the regression measure presented, the deep learning model performed only 0.039 better than the baseline. However, observing the confusion matrix on figure 19 it can be seen that the values are concentrated around the diagonal. The MSE of the Deep Learning model is 180.32, yielding a mean error of 13.42 - thus, on average, the model predicts the age correctly, within 13.42 years. As can be seen in the confusion matrix, the model is fairly accurate, for customers in the range 18-28, and likewise for customers in age range 78-84.

11.2 Task 2

This task is concerned with performing binary classification of customers on the range 18-27 and 28-84.

	Training	Test	Validation	Sampling method
Deep Learning Model	586028	451431	501590	Balanced
Random Forest	834372	147243	-	Unbalanced

Table 14: Task 2, number of data points used, and sampling method.

Figure 14 shows the number of data points used, together with sampling method.

	RandomForest	Deep Learning	Baseline
BOW	0.3636	0.3955	0.0721
Deep Features	0.3729	0.4074	0.0721
Word2vec	0.3569	0.3405	0.0721

Table 15: Task 2 results

Figure 15 shows the results of the experiment. The deep learning model, with the deep features performed the best, directly followed by deep learning using BOW. Both models performed significantly better than the baseline.

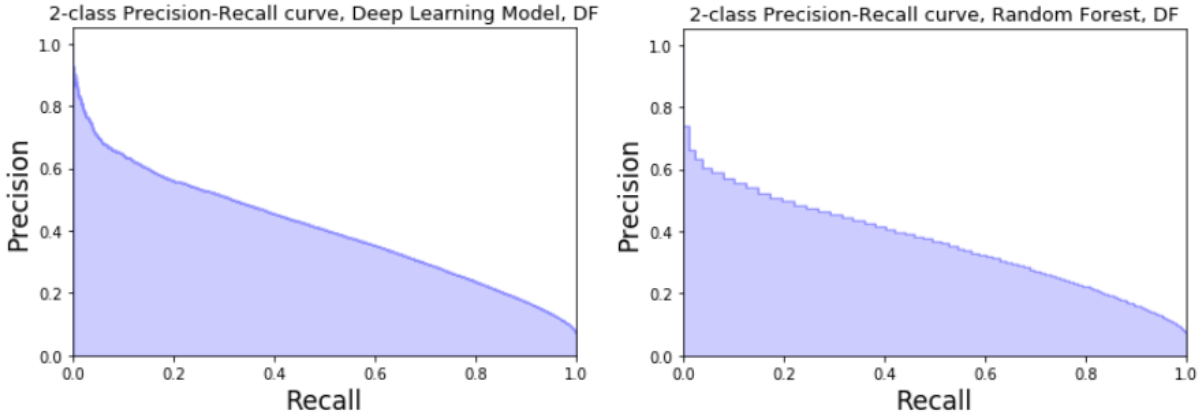

Figure 20: Precision-Recall curve for the deep learning model using the deep learning features on the left, random forest using the deep learning features on the right.

Figure 20 shows the Precision-Recall curve for each of the models with the highest performing features. The

plots are similar. The deep learning model has a higher precision, when recall is lower than 0.1, than the random forest model.

11.3 Task 3

Here we predict if visitor clicks on the "*Become a customer*" link.

	Training	Test	Validation	Sampling method
Deep Learning Model	255894	28433	31592	Unbalanced
Random Forest	287486	28433	-	Unbalanced

Table 16: Task 3, number of data points used, and sampling method.

Figure 16 shows the number of data points used for the task, together with sampling method. There was a total of 7500 samples where people has clicked "Become a customer" in the training dataset, 941 in the validation set, and 821 in the test set. The ratio between the positive and negative class is ca. 33 to 1 - the dataset is highly unbalanced. To mitigate this for the deep learning model, weight for the classes was used - class 1 has a weight of 33, and class 0 has a weight of 1. The class weights applies strictly to the training phase, where the loss of sample with label 1 counts 33 as much, as a loss for a sample with class 0.

	RandomForest	Deep Learning	Baseline
BOW	0.70459	0.8803	0.0333
Deep Features	0.9004	0.8927	0.0333
Word2vec	0.8188	0.7974	0.0333

Table 17: Task 3 results.

Figure 17 shows the performance of the models. The best performing model was random forest using the deep learning features. It had an PR-AUC of over 0.90.

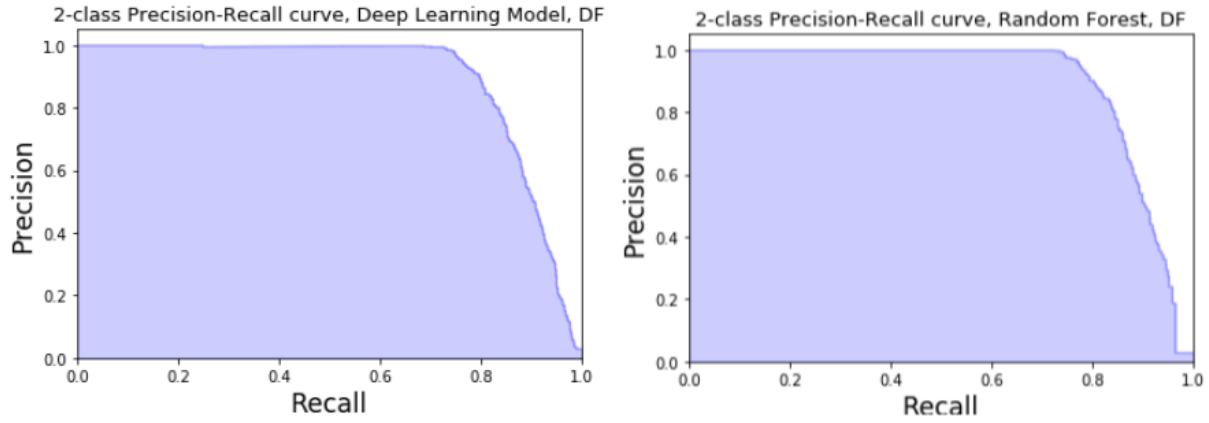

Figure 21: Precision-Recall curve for the deep learning model and random forest, using the deep features.

Figure 21 shows the Precision-Recall curve for the deep learning , and random forest model, using deep learning features. Both are very close to 1, until recall is around 0.78. The figure shows, that if we were to predict if a customer is going to click "Become a customer" then the random forest model could potentially (by setting the corresponding threshold value) be correct very close to 100% in all of its predictions, for 77% of the people who clicks become a customer.

12 Discussion

In this section we will discuss the results presented in the previous section. Here we will explain the results which we obtained and whether we successfully completed the given tasks.

For task 1, we see from table 13, that the models were barely able to beat the baseline, which is surprising because as we saw in the description of the task, there appeared to be a reasonable correlation between the features and ages of visitors. It is also clear from table 13 that the representation of the click path had a relatively small effect on the results. Here we expected Word2Vec and the learned representation of the clicks to perform substantially better than the Bag of Words representation, as the Bag of Words representation does not regard the context of the clicks. Looking at the confusion matrices, shown for task 1 in figure 19, the predictions tends towards the diagonal for both plots, which is desired as this indicates correct predictions across all targets. The predictions are however not precise enough to give a good performance on the chosen evaluation function. As evident from the leftmost confusion matrix, showing the predictions for the deep learning model, the predictions deviate a great deal around the diagonal, causing many predictions to lie outside the ± 5 age interval. The right confusion matrix is more confined, however it does not lie perfectly on the diagonal and we see that the majority of predictions made for ages greater than 75 are predicted in the age interval 65-71. Thus we can see that the predictions were not completely random and that the models were able to predict the ages to a certain degree, however not precise enough to yield a satisfactory result on the evaluation function.

For task 2 we were able to beat the baseline, however as evident from table 15 the AUC scores were relatively low. Figure 20 shows the precision as a function of recall i.e. the precision-recall curve, for the Random Forest and deep learning classifier. For both plots, the precision decreases rapidly as recall increases. Thus, even when we only take into account predictions, which the classifier with high probability predicted to be in the age group 18-27, there still occurs a great deal of misclassifications. As for task 1, we see that the click representation has little to no impact on the result.

Our initial hypothesis, for the age tasks, was that we would be able to successfully predict the ages of visitors. While getting somewhat meaningful results, the models were not precise enough to achieve satisfactory results. We believe that one of the main reasons for the subpar results is a lacking correlation between the click paths and the ages of visitors. Through the Random Forest models, we extracted feature importances, for both tasks to inspect how important the clicks were to the predictions. We also take the feature importances of each representation of the clicks, to see if some representation lead to higher importances than other. To do so, the feature importances were divided into two groups: the click path representation and the rest of the features. These feature importances are shown in table 18 and 19, which shows the feature importances for task 1 and 2 respectively.

	Word2Vec	Bag of Words	Learned representation
Click	0.356	0.283	0.571
Rest	0.643	0.716	0.428

Table 18: Relative feature importances for task 1.

	Word2Vec	Bag of Words	Learned representation
Click	0.359	0.304	0.640
Rest	0.640	0.695	0.359

Table 19: Relative feature importances for task 2.

As evident from table 18 and 19. The click importances are similar for the two tasks. The learned representation has a higher relative importance, which is also the representation with the best results over the two tasks. The tables stem well with our previous hypothesis that the representations, which take into account the context of the clicks, will have better performance on the tasks.

For task 3, we were able to achieve better performance over the two models and across the different representations of the clicks. As evident from table 17, the best results was achieved using Random Forest with the learned representation of the clicks, closely followed by the deep learning model, using the same representation. For Random Forest, the Bag of Words representation achieved the worst results, while the Word2Vec representation gave significantly better results. This is most likely due to the Bag of Words representation being sparse, which is not handled well by the Random Forest model. The opposite pattern was observed for the deep learning model, where the Bag of Words representation outperformed the Word2Vec representation. Figure 21 shows the Precision-recall curves for the Deep learning and Random Forest models. As evident from the figure, the Precision remains at 1, for the most part of the plot, thus classifying all predictions made on the positive class correctly. The precision starts to decrease around recall = 0.77 and thus the model is able to predict a great deal of the positive class, without producing a single misclassification on the class. Table 20 shows the relative feature importance between the click representation and the rest of the features. The click representation is significantly more important for this task, compared to the two previous tasks. Here we see that for all representations, the clicks were valued more important than the rest of the of the features, especially for the learned features, where the representation make out 90% of the feature importance for the task.

	Word2Vec	Bag of Words	Learned representation
Click	0.746	0.601	0.903
Rest	0.253	0.398	0.096

Table 20: Relative feature importances for task 3.

Figure 22 shows how the click features for task 3 looks, as they are projected down to a 2D and 3D space, using PCA. the x, y, and z axis explains a ratio of 0.658, 0.263, 0.043, of the total variance, respectively.

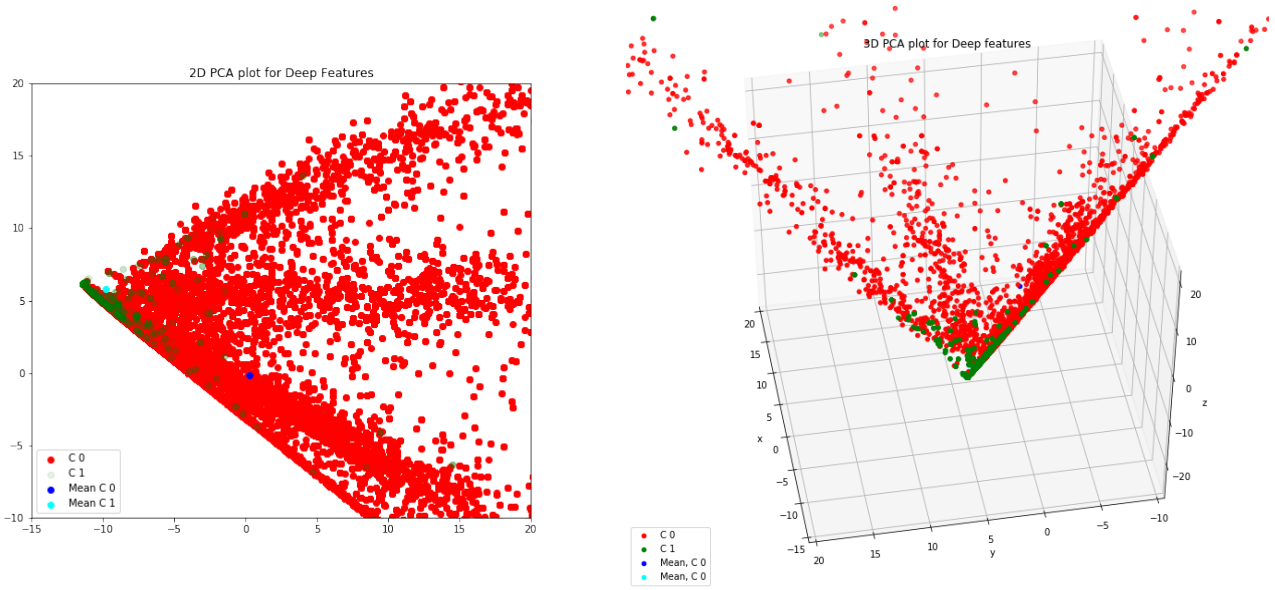

Figure 22: 2D- and 3D PCA plot, for task 3. Red is click features of people who did not click login in (C 0), and green is the click features of people who did (C 1). The cyan dot shows the mean of class 0, and the blue shows the mean of class 1.

As can be seen, class 1 is clustered along a line, and the points for this class are relatively close to the mean. The points for class 0 appears to be spread out, along 3 lines.

For these experiments, we have considered the entire click paths of each visitor. In practice however, a decision needs to be made within a few clicks, for the decision to have any effect on the visitor and to be of value to the bank. Therefore we have performed the experiment at different cut-off's in number of clicks, i.e. limited each session to a variable number of clicks, in order to see how many clicks the model must observe, before outputting a good prediction. These experiments have been performed on task 3, with the most successful model and representation of the features, i.e. Random Forest with the learned representation of clicks. Figure 23 shows the AUC score as a function of the number of clicks observed by the model.

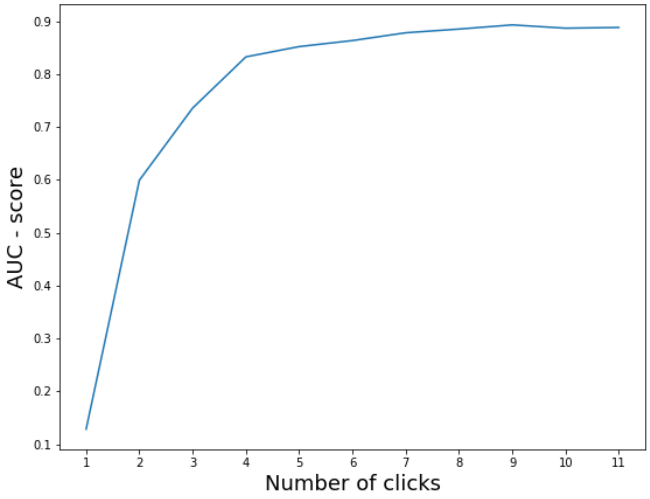

Figure 23: Representation of time between clicks.

As evident from figure 23, taking into account only very few clicks results in bad results. The AUC score increases rapidly as the number of observed clicks increases and then flattens out after around 4 to 5 clicks. After 4 clicks, the AUC is 0.832, which is not as good as considering the entire click path, but relatively good and could be used as a reasonable cut-off, when used in practice.

13 Conclusion

We have outlined the Celebrus data, and described the relevant tables for the given tasks. We have shown a generic modular pipeline, capable of processing the Celebrus data within the banks ecosystem. A pipeline, which we used both for data cleaning and feature extraction. We have tried a wide variety of traditional machine learning models and ended up using Random Forest and multilayer perceptron, as these performed the best on the three tasks. From these, we found that there was no single model, which outperformed the other, however overall the Neural networks approach gave slightly better results. We can conclude that while not being able to solve task 1 and 2 successfully, we were able to solve task 3. Therefore we can conclude that the Celebrus data can indeed be used to model visitor behaviours. We can also conclude that using a click representation, which takes into account the context of the clicks, gives better results. Here we also saw that the learned representation gave the best results.

13.1 Future work

Throughout this project, we have worked on solving the tasks, using only the Celebrus data. The main purpose, of the implementation of the Celebrus data in the bank, is to use the data to facilitate the existing models, this could for example be to assess whether or not a customer will leave the bank. These models will most likely require using Celebrus data across sessions, i.e. tracking customer behaviour over different sessions, whereas we have based our analysis on individual sessions. Thus the next step would be to analyse customer behaviour over different sessions. One area, which the Celebrus data alone has a wide array of applications is within marketing. Thus new tasks could be set up in correlation with the marketing department of Danske Bank.

14 Appendix

14.1 Celebrus table definition

Listing 2:

```
1  -- -----------------------------------------------------
2  -- Table `DataLoader`.`Visitor`
3  -- -----------------------------------------------------
4  CREATE TABLE IF NOT EXISTS `DataLoader`.`Visitor` (
5  `TrackingUuid` BINARY(16) NOT NULL COMMENT 'A unique identifier assigned to
       one data file',
6  `UVTFrequency` INT(11) NULL DEFAULT NULL COMMENT 'Number of times this user
       has visited with this uvt cookie.',
7  `UVTPreviousSessionTimestamp` TIMESTAMP(3) NULL COMMENT 'The timestamp of the
       users previous session number, if known/exists',
8  `UVTPreviousSessionNumber` BIGINT(20) NULL DEFAULT NULL COMMENT 'The users
       previous session number, if known/exists',
9  `UVTIsNewAcquisition` TINYINT(4) NULL DEFAULT NULL COMMENT 'True, if this
       session was the first for the user on this uvt cookie.',
10 `UVTAcquisitionTimestamp` TIMESTAMP(3) NULL DEFAULT NULL COMMENT 'Timestamp of
        the users first session on this uvt cookie.',
11 `ProfileUiid` VARCHAR(250) NULL DEFAULT NULL COMMENT 'Unique individual
       identifier for the user. This should be a unique login name, email address
       , or similar. It (along with cookie uvt) permits tracking the actions of
       users across multiple sessions.',
12 `CookieUniqueVisitorTrackingId` VARCHAR(1000) NULL COMMENT 'The unique uvt
       cookie id of the user, taken from the full uvt cookie.',
13 `SessionNumber` BIGINT(20) NOT NULL COMMENT 'Unique number assigned to the
       session',
14 `EventID` BINARY(16) NOT NULL COMMENT 'Unique identifier of this specific
       event',
15 `EventTimestamp` TIMESTAMP(3) NOT NULL COMMENT 'Server timestamp at which the
       event/action associated with the event occurred. For derived events this
       is normally equal to the timestamp of the event that triggered graph
       completion.',
16 `IdSequenceNumber` INT(11) NOT NULL COMMENT 'A sequence number for user
       identifications throughout the session (1st index being \'1\').',
17 PRIMARY KEY (`SessionNumber`, `EventID`),
18 INDEX `idx_VisitorTimestamp` (`EventTimestamp` ASC),
19 CONSTRAINT `fk_Visitor_SStart`
20 FOREIGN KEY (`SessionNumber`)
21 REFERENCES `DataLoader`.`SessionStart` (`SessionNumber`)
22 ON DELETE NO ACTION
23 ON UPDATE NO ACTION)
24 ENGINE = InnoDB
25 AVG_ROW_LENGTH = 512
26 COMMENT = 'Details of visitors to the site. One row per combination of UVT and
        Uiid seen within a given Session.'
27 MAX_ROWS = 104857600;
```

```
28
29 -- -----------------------------------------------------------
30 -- Table `DataLoader`.`SessionStart`
31 -- -----------------------------------------------------------
32 CREATE TABLE IF NOT EXISTS `DataLoader`.`SessionStart` (
33 `TrackingUuid` BINARY(16) NOT NULL COMMENT 'A unique identifier assigned to
       one data file',
34 `DeviceIPAddress` VARCHAR(50) NULL DEFAULT NULL COMMENT 'IP Address browsed
       from',
35 `DeviceUserAgent` VARCHAR(250) NULL DEFAULT NULL COMMENT 'The full browser
       user agent',
36 `DeviceBrowserName` VARCHAR(250) NULL DEFAULT NULL COMMENT 'Name of the users
       browser',
37 `DeviceBrowserVersion` VARCHAR(250) NULL DEFAULT NULL COMMENT 'Version number
       of the users browser',
38 `DeviceCookiesEnabled` TINYINT(4) NULL DEFAULT NULL COMMENT 'True/False - are
       permanent cookies enabled on the browsing system',
39 `DevicePlatformName` VARCHAR(250) NULL DEFAULT NULL COMMENT 'Operating system
       of the user',
40 `DevicePlatformGroup` VARCHAR(250) NULL COMMENT 'A summarised grouping of
       operating system - e.g. Windows, Unix/Linux',C
41 `DeviceSystemName` VARCHAR(250) NULL COMMENT 'The system on which the browser
       is running. e.g. pc, mobile phone make, specific games console',
42 `DeviceSystemType` VARCHAR(250) NULL COMMENT 'A broad grouping of system split
        into mobile, pc or console',
43 `SessionPartialOptOut` TINYINT(4) NULL DEFAULT NULL COMMENT 'True, if the user
        chose to partially opt-out of collection via P3P',
44 `SessionConfiguredTimeOut` BIGINT(20) NULL DEFAULT NULL COMMENT 'The
       configured session timeout in milliseconds, as set in the collection
       server.',
45 `SessionIsContinuation` TINYINT(4) NULL DEFAULT NULL COMMENT 'Is this session
       the continuation of a previous one which timed out',
46 `LinkedSessionNumber` BIGINT(20) NULL DEFAULT NULL COMMENT 'The session number
        of the previously timed out session, if this session is the continuation
       of a one which timed out',
47 `LinkedSessionKey` VARCHAR(250) NULL DEFAULT NULL,
48 `SessionNumber` BIGINT(20) NOT NULL COMMENT 'Unique number assigned to the
       session',
49 `SessionKey` VARCHAR(250) NULL DEFAULT NULL COMMENT 'Key used to ensure
       session security in processing and real time queries/rules',
50 `EventTimestamp` TIMESTAMP(3) NOT NULL COMMENT 'Server timestamp at which the
       event/action associated with the event occurred. For derived events this
       is normally equal to the timestamp of the event that triggered graph
       completion.',
51 `SessionBaseCurrencyCode` CHAR(3) NOT NULL COMMENT 'Base ISO Currency code for
        all values within the session. This currency is the currency which the
       system will have converted the source values/currencies into to from any
       base values.',
```

```
52 `SessionDataSourceName` VARCHAR(75) NULL DEFAULT NULL COMMENT 'Identifies the
       source of this Celebrus session. Can be either \"Celebrus Web\" or \"
       Celebrus Mobile App\".',
53 PRIMARY KEY (`SessionNumber`),
54 INDEX `idx_SStartTimestamp` (`EventTimestamp` ASC))
55 ENGINE = InnoDB
56 AVG_ROW_LENGTH = 512
57 COMMENT = 'Indicates the start of a session; one row per session.'
58 MAX_ROWS = 104857600;
59
60 -- -----------------------------------------------------
61 -- Table `DataLoader`.`Page`
62 -- -----------------------------------------------------
63 CREATE TABLE IF NOT EXISTS `DataLoader`.`Page` (
64 `TrackingUuid` BINARY(16) NOT NULL COMMENT 'A unique identifier assigned to
       one data file',
65 `PageLocationDomain` VARCHAR(1000) NULL DEFAULT NULL COMMENT 'Url domain of
       the page',
66 `PageTitle` VARCHAR(1000) NULL DEFAULT NULL COMMENT 'Title of the page as
       displayed in the browser.',
67 `ReferringPageInstanceID` BIGINT(20) NULL DEFAULT NULL COMMENT 'The CSA number
        of the page',
68 `SessionNumber` BIGINT(20) NOT NULL COMMENT 'Unique number assigned to the
       session',
69 `PageInstanceID` BIGINT(20) NOT NULL COMMENT 'The CSA number of the page',
70 `EventTimestamp` TIMESTAMP(3) NOT NULL COMMENT 'Server timestamp at which the
       event/action associated with the event occurred. For derived events this
       is normally equal to the timestamp of the event that triggered graph
       completion.',
71 `PageLocation` VARCHAR(1000) NULL DEFAULT NULL COMMENT 'Full URL of the page',
72 `AttributionSequenceInSession` INT(11) NULL DEFAULT NULL COMMENT 'Sequence
       identifier for this campaign attribution within the session. For example,
       is it the first attribution, or fifth.',
73 `PageSequenceInSession` INT(11) NULL DEFAULT NULL COMMENT 'Sequence number of
       the page within the session',
74 `PageSequenceInAttribution` INT(11) NULL DEFAULT NULL COMMENT 'Sequence number
        of the page since the most recent campaign attribution',
75 `TopLevelWindowID` VARCHAR(1000) NULL DEFAULT NULL COMMENT 'Unique ID of the
       top level window of the browser (browsers with multiple windows/tabs will
       have different top level window ids for each window/tab).',
76 `ProfileUiid` VARCHAR(250) NULL COMMENT 'Latest observed unique individual
       identifier for the user. This is generally a unique login name, email
       address, or similar. It (along with cookie uvt) permits tracking the
       actions of users across multiple sessions.',
77 PRIMARY KEY (`SessionNumber`, `PageInstanceID`),
78 INDEX `idx_PageTimestamp` (`EventTimestamp` ASC),
79 CONSTRAINT `fk_Page_SStart`
80 FOREIGN KEY (`SessionNumber`)
81 REFERENCES `DataLoader`.`SessionStart` (`SessionNumber`)
```

```
82 ON DELETE NO ACTION
83 ON UPDATE NO ACTION)
84 ENGINE = InnoDB
85 AVG_ROW_LENGTH = 512
86 COMMENT = 'Information on a given page populated at the time of loading. One
       row per page. Information which builds up over the lifetime of a page (e.g
       . view times) are included in PageSummary.'
87 MAX_ROWS = 104857600;
88
89 -- -----------------------------------------------------
90 -- Table `DataLoader`.`Click`
91 -- -----------------------------------------------------
92 CREATE TABLE IF NOT EXISTS `DataLoader`.`Click` (
93 `TrackingUuid` BINARY(16) NOT NULL COMMENT 'A unique identifier assigned to
       one data file',
94 `ObjectAlttext` VARCHAR(1000) NULL DEFAULT NULL COMMENT 'Alt text string
       defined within the html, for the given object',
95 `UnformattedHierarchyName` VARCHAR(1000) NULL DEFAULT NULL COMMENT 'Full html-
       hierarchical name of the object, without any normalisation - used for
       overlays',
96 `UnformattedHierarchyID` VARCHAR(1000) NULL DEFAULT NULL COMMENT 'Full html-
       hierarchical id of the object, without any normalisation - used for
       overlays',
97 `ObjectHierarchyName` VARCHAR(1000) NULL DEFAULT NULL COMMENT 'Full html-
       hierarchical name of the object, normalised to remove excess delimiters (
       format: parentXS/parentX/child)',
98 `ObjectHierarchyID` VARCHAR(1000) NULL DEFAULT NULL COMMENT 'Full html-
       hierarchical id of the object, normalised to remove excess delimiters (
       format: parentXS/parentX/child)',
99 `ObjectSrc` VARCHAR(1000) NULL DEFAULT NULL COMMENT 'Source href of the object
       , if such exists (e.g. for images)',
100 `ObjectIsLink` TINYINT(4) NULL DEFAULT NULL COMMENT 'True, if the object
       represents an anchor/link',
101 `ObjectIsDownload` TINYINT(4) NULL DEFAULT NULL COMMENT 'True, if the object
       matches the configuration for what constitutes a download.',
102 `ObjectType` VARCHAR(1000) NULL DEFAULT NULL COMMENT 'The enum name of the
       ClickObjectTypeEnum object which catagorises the type of object this is.
       Output only used by IWI. Note this is different from ObjectSourceType',
103 `ObjectSourceType` VARCHAR(1000) NULL DEFAULT NULL COMMENT 'The trapped event
       property object_source_type. This can be user defined within the web page
       and is used solely by magiq configuration definitions.',
104 `ObjectTagName` VARCHAR(1000) NULL DEFAULT NULL COMMENT 'Tag name of the
       object within the html',
105 `FieldIsChainedItemSelect` TINYINT(4) NULL DEFAULT NULL COMMENT 'Is the
       interaction on the field the 2nd or subsequent item select in a continuous
        stream of item selects. This is used to handle scrolling through drop-
       down lists or similar which yield multiple changes before settling on a
       value.',
```

```
106 `PreviousChainedISEventIndex` INT(11) NULL DEFAULT NULL COMMENT 'If the object
        is a chained item select, then this is the event index of the item select
        this one is chained from.',
107 `FormName` VARCHAR(250) NULL DEFAULT NULL COMMENT '(Non-Orphaned Forms Only):
        The html name of the form',
108 `FormID` VARCHAR(1000) NULL DEFAULT NULL COMMENT '(Non-Orphaned Forms Only):
        The html id of the form',
109 `ObjectHref` VARCHAR(1000) NULL DEFAULT NULL COMMENT 'Anchor href of the
        object, if such exists. If not, then this is populated with the source
        href.',
110 `ObjectID` VARCHAR(1000) NULL DEFAULT NULL COMMENT 'HTML id of the object',
111 `ObjectClass` VARCHAR(1000) NULL DEFAULT NULL COMMENT 'Html-class of the
        object',
112 `ObjectName` VARCHAR(1000) NULL DEFAULT NULL COMMENT 'Html-name of the object
        ',
113 `SessionNumber` BIGINT(20) NOT NULL COMMENT 'Unique number assigned to the
        session',
114 `EventTimestamp` TIMESTAMP(3) NOT NULL COMMENT 'Server timestamp at which the
        event/action associated with the event occurred. For derived events this
        is normally equal to the timestamp of the event that triggered graph
        completion.',
115 `EventID` BINARY(16) NOT NULL COMMENT 'Unique identifier of this specific
        event',
116 `PageInstanceID` BIGINT(20) NOT NULL COMMENT 'The CSA number of the page',
117 `PageEventIndex` INT(11) NOT NULL COMMENT 'Event index of the base trapped
        event that resulted in the creation of this event. For derived events this
        is pulled from the event that resulted in the derived events creation.',
118 PRIMARY KEY (`SessionNumber`, `PageInstanceID`, `PageEventIndex`),
119 INDEX `idx_Click` (`SessionNumber` ASC, `EventID` ASC),
120 INDEX `idx_ClickTimestamp` (`EventTimestamp` ASC),
121 CONSTRAINT `fk_Click_Page`
122 FOREIGN KEY (`SessionNumber` , `PageInstanceID`)
123 REFERENCES `DataLoader`.`Page` (`SessionNumber` , `PageInstanceID`)
124 ON DELETE NO ACTION
125 ON UPDATE NO ACTION)
126 ENGINE = InnoDB
127 AVG_ROW_LENGTH = 512
128 COMMENT = 'A click/interaction with an object on the website (e.g. buttons,
        links, text boxes or similar). One row per interaction.'
129 MAX_ROWS = 104857600;
```

14.2 Project database

Listing 3:

```
1 Table name                                Row count
2 ------------------------------------------------
3 hd_wt_behaviour                           0
4 hd_wt_clickalias                          0
```

5	hd_wt_contentaction	0
6	hd_wt_contentclickthru	0
7	hd_wt_customattributes	0
8	hd_wt_downloadalias	0
9	hd_wt_formsuccess	0
10	hd_wt_individualattribute	0
11	hd_wt_individualemail	0
12	hd_wt_individualmailingaddress	0
13	hd_wt_individualtelephone	0
14	hd_wt_media	0
15	hd_wt_mediaalias	0
16	hd_wt_mediamarker	0
17	hd_wt_optedout	0
18	hd_wt_pageerrorattributes	0
19	hd_wt_pathsegment	0
20	hd_wt_promotion	0
21	hd_wt_sessionclassification	0
22	hd_wt_transactioncomplete	0
23	hd_wt_transactioncompleteattrb	0
24	hd_wt_transactionstepattrb	0
25	hd_wt_triggeraction	0
26	hd_wt_transactionabandoned	1
27	hd_wt_transactionstart	1
28	hd_wt_transactionstep	1
29	hd_wt_metricvalue	10
30	hd_wt_download	232357
31	hd_wt_internalsearchterm	491208
32	hd_wt_internalsearch	496724
33	hd_wt_internalsearchoutcome	694670
34	hd_wt_goal	1725538
35	hd_wt_formsubmit	22513056
36	hd_wt_individual	23973383
37	hd_wt_pageerror	28026873
38	hd_wt_fieldsubmit	32146833
39	hd_wt_geolocation	43969262
40	hd_wt_sessionstart	44487594
41	hd_wt_sessionsummary	44846154
42	hd_wt_visitorsummary	46632079
43	hd_wt_campaignattributionstart	48639530
44	hd_wt_campaignattributionend	49353279
45	hd_wt_fieldidentification	68480020
46	hd_wt_formidentification	73266904
47	hd_wt_visitor	81556408
48	hd_wt_fieldinteraction	132913401
49	hd_wt_customerjourney	133701391
50	hd_wt_pagesummary	226983271
51	hd_wt_page	330980849
52	hd_wt_click	457007996
53	hd_wt_pagealias	504194957

14.3 Deep learning model

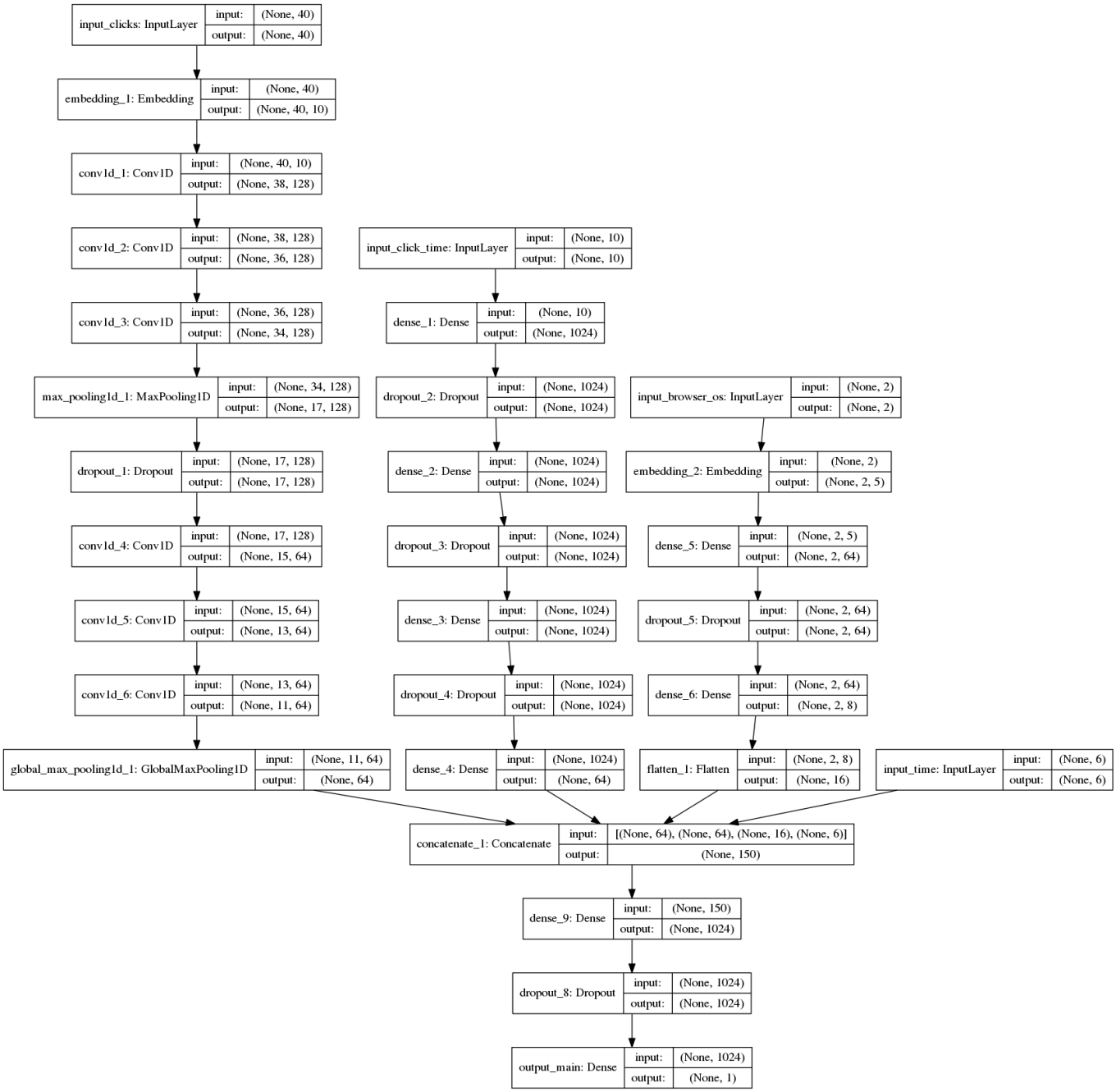

Figure 24: Model architecture, as generated by Keras.

Output from keras:

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_clicks (InputLayer)    (None, 40)                0
```

embedding_1 (Embedding)	(None, 40, 10)	4950	input_clicks[0][0]
conv1d_1 (Conv1D)	(None, 38, 128)	3968	embedding_1[0][0]
conv1d_2 (Conv1D)	(None, 36, 128)	49280	conv1d_1[0][0]
input_click_time (InputLayer)	(None, 10)	0	
conv1d_3 (Conv1D)	(None, 34, 128)	49280	conv1d_2[0][0]
dense_1 (Dense)	(None, 1024)	11264	input_click_time[0][0]
max_pooling1d_1 (MaxPooling1D)	(None, 17, 128)	0	conv1d_3[0][0]
dropout_2 (Dropout)	(None, 1024)	0	dense_1[0][0]
input_browser_os (InputLayer)	(None, 2)	0	
dropout_1 (Dropout)	(None, 17, 128)	0	max_pooling1d_1[0][0]
dense_2 (Dense)	(None, 1024)	1049600	dropout_2[0][0]
embedding_2 (Embedding)	(None, 2, 5)	75	input_browser_os[0][0]
conv1d_4 (Conv1D)	(None, 15, 64)	24640	dropout_1[0][0]
dropout_3 (Dropout)	(None, 1024)	0	dense_2[0][0]
dense_5 (Dense)	(None, 2, 64)	384	embedding_2[0][0]
conv1d_5 (Conv1D)	(None, 13, 64)	12352	conv1d_4[0][0]
dense_3 (Dense)	(None, 1024)	1049600	dropout_3[0][0]
dropout_5 (Dropout)	(None, 2, 64)	0	dense_5[0][0]
conv1d_6 (Conv1D)	(None, 11, 64)	12352	conv1d_5[0][0]
dropout_4 (Dropout)	(None, 1024)	0	dense_3[0][0]
dense_6 (Dense)	(None, 2, 8)	520	dropout_5[0][0]
global_max_pooling1d_1 (GlobalM	(None, 64)	0	conv1d_6[0][0]
dense_4 (Dense)	(None, 64)	65600	dropout_4[0][0]
flatten_1 (Flatten)	(None, 16)	0	dense_6[0][0]
input_time (InputLayer)	(None, 6)	0	

```
_____________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 150)          0           global_max_pooling1d_1[0]
dense_4[0][0]
flatten_1[0][0]
input_time[0][0]
_____________________________________________________________________________________________
dense_9 (Dense)                 (None, 1024)         154624      concatenate_1[0][0]
_____________________________________________________________________________________________
dropout_8 (Dropout)             (None, 1024)         0           dense_9[0][0]
_____________________________________________________________________________________________
output_main (Dense)             (None, 1)            1025        dropout_8[0][0]
=============================================================================================
Total params: 2,489,514
Trainable params: 2,489,514
Non-trainable params: 0
```

References

[1] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[2] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

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

[4] François Chollet et al. Keras. `https://keras.io`, 2015.

[5] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.