Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

User Behavior Clustering and Behavior Modeling Based on Clickstream Data

Jan Žaloudek

Supervisor: Ing. Jan Drchal, Ph.D.
May 2018

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Žaloudek** Jméno: **Jan** Osobní číslo: **393687**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Shlukování a modelování chování uživatelů založené na datech z webového prohlížeče

Název diplomové práce anglicky:

User Behavior Clustering and Behavior Modeling Based on Clickstream Data

Pokyny pro vypracování:

Study the current state of the art of user behavior modeling and clustering.Design a behavior clustering algorithm. Consider extensions to semi-supervised learning.Train models using data provided by Smartlook. Experiment with features beyond URL addresses and mouse clicks such as web browser traces. Evaluate the quality of all models.

Seznam doporučené literatury:

[1] Jonáš Amrich. Modeling on-line user behavior using url embedding, diploma thesis, CTU, 2017.
[2] Igor Cadez, David Heckerman, Chris Meek, Padhraic Smyth, and Steven White. Visualization of navigation patterns on a web site using model based clustering. Technical report, March 2000.
[3] Michael Scholz. R package clickstream: Analyzing clickstream data with markov chains. Journal of Statistical Software, Articles, 74(4):1?17, 2016.
[4] Gang Wang, Xinyi Zhang, Shiliang Tang, Haitao Zheng, and Ben Y. Zhao. Unsupervised clickstream clustering for user behavior analysis. In Proceed- ings of the 2016 CHI Conference on Human Factors in Computing Systems, CHI ?16, pages 225?236, New York, NY, USA, 2016. ACM.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Jan Drchal, Ph.D., centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **02.03.2018** Termín odevzdání diplomové práce: _____

Platnost zadání diplomové práce: **30.09.2019**

Ing. Jan Drchal, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____ Datum převzetí zadání

_____ Podpis studenta

Acknowledgements

I would like to thank my supervisor Jan Drchal for guidance and patience help during my work on this thesis. I would also like to thank Smartlook.com for providing the dataset.

Special thanks belong to my family, girlfriend, and friends for great support during the time of my studies and writing of this thesis.

Declaration

I hereby declare that I worked out the presented thesis independently and I quoted all the sources used in this thesis in accord with Methodical instructions about ethical principles for writing academic thesis.

In Prague, . May 2018

.....

Jan Žaloudek

Abstract

With growing online population having a good understanding of users' behavior on the internet is becoming very important. In this thesis, we explore different ways how to represent the behavior of online users and how to apply clustering and semi-supervised learning methods to this knowledge.

We propose two different approaches how to transform captured clickstreams together with other events to the vector space. The first method is based on frequencies of the events and the second one is build on top of distributed bag-of-words. The second method shows promising results in clustering and semi-supervised tasks.

Keywords: Web usage mining, User behavior clustering, Clickstream modeling, Distributed representation of documents, Semi-supervised learning, Machine learning

Supervisor: Ing. Jan Drchal, Ph.D.

Abstrakt

S rostoucím počtem uživatelů internetu je dobrá znalost chování jeho uživatelů nezbytná. V této práci zkoumáme různé způsoby, jak reprezentovat chování uživatelů internetu a možnosti, jak tuto znalost poté použít pro shlukování a pro částečně řízené učení.

Navrhujeme dva různé způsoby transformace snímaných clickstreamů společně s dalšími událostmi z prohlížeče do vektorového prostoru. První metoda je založena na frekvencích událostí a druhá je založena na distribuovaném bag-of-words. Druhá představená metoda vykazuje slibné výsledky při úkolech shlukování a částečně řízeného učení.

Klíčová slova: Web usage mining, Shlukování chování uživatelů, Modelování clickstreamu, Distribuovaná reprezentace dokumentů, Částečně řízené učení, Strojové učení

Překlad názvu: Shlukování a modelování chování uživatelů založené na datech z webového prohlížeče

Contents

Figures

Tables

Chapter 1

Introduction

Discovering human behavioral patterns is topic studied by many academic disciplines such as psychology, sociology and now even computer science. Understanding human behavior on the internet is becoming a big part of understanding modern society, where people spend online most of their lives. There is an option to do surveys and directly ask people about their opinions; however thanks to the technological progress it is now possible to collect more data than ever, and with all the computational power we have today, it is possible to dive deep to those data and start discovering new patterns.

The behavior of the users on the internet can be recorded in many different formats; however, in the end, the user behavior is usually captured as a clickstream. Along with the clickstream, which is a log of all clicks that user performed in the browser, other information may be stored. That extra information can be values in forms, scrolling in a page, Javascript errors or even custom events from the application itself. As applications nowadays getting more complex, clicking on some element often does lead to change of the actual URL, but instead, it switches some internal state of the application in the browser. Therefore following only URLs seems not to be sufficient for complex behavior analysis in modern application.

Clickstream analysis is from the group of Web Mining tasks - more precisely Web Usage Mining. There are many fields where knowledge of user's online behavior can be applied and valued. There are application designers who want to know, how the users interact with the user interface. There is also lot of work aiming at security and frauds detection. Worth mentioning is the use of those data in e-commerce, where sellers can predict groups of their customers, and better target their products using some kind of recommender system. There are different approaches, how to represent actual user behavior and how to find a relation between two users. In this thesis we will focus on a close relation between modeling user behavior and natural laguage processing, so called NLP, tasks.

1.1 Goals

There are three primary goals in this thesis. First of all, there is a need for a user behavior representation. As we mentioned in the introduction, there is a close relation between modeling an user behavior and modeling text documents for NLP tasks, therefore we utilize some of already well-known techniques from NLP in our work. Next task is to clusterize representation obtained in the first step, and the final task is to propose a semi-supervised extension.

1.1.1 User behavior representation

At first, it is necessary to find a way, how to represent a user behavior. There are several options mentioned in Chapter 2. However, clickstream data are usually very noisy, and it can be tricky to find ideal representation. Next step is to somehow incorporate other data that are available in the dataset (such as browser errors, changes of URLs or typing into inputs) into that representation. Construction of solid representation is crucial for other tasks that will take use of that - such as clustering or classification, to obtain some quality results.

1.1.2 Clustering behavior models

After obtaining a suitable representation of user behavior model, next task is to clusterize users into somehow similar groups. There are already many different clustering algorithms, where each of them is aiming at a different aspect of data being clusterized. The task will be to investigate current state-of-the-art options and maybe propose an extension to the existing solution. Clustering can be highly subjective when it comes to interpretation of the results. Therefore it is not easy to evaluate obtained cluster assignments for the data. This issue can be diminished by investigating options provided by semi-supervised learning.

1.1.3 Semi-supervised extension

It is common for users' clickstream data that there is some extra information available about the users. This information can be used for training semi-supervised models. Those models can be used for example for suspicious users detection, where usually only just a small portion of labeled users is known and unlabeled portion of the users is much higher.

Chapter 2

Related Work

In past years many approaches to modeling and clustering user behavior emerged. How web applications evolved and growing population of internet users grew and became more demanding, the complexity of web application grew as well. This chapter is divided into three sections. We start with research of related problems in NLP and their connection with user behavior modeling. Then we examine methods based on Markov chains. After that we discuss some related work for clustering and semi-supervised learning.

2.1 Behavior modeling

2.1.1 Representation with events frequencies

The most straightforward approach to user behavior representation is done by counting events occurrences. Despite its simplicity it is widely used in different applications of clickstream analysis and behavior modeling. It can be seen applied in [Wang et al., 2016] [Wei et al., 2012] [Wang et al., 2017]. This method can be seen as the variation of Bag of Words method mentioned in Section 3.1, which was proposed by Harris [Harris, 1954] in 1954 and then extended by Jones [Jones, 1973] in 1973. Similarity between BoW method, that is used in NLP, can imply a close connection between NLP and clickstream modeling.

2.1.2 Neural network based methods

Very recent proposed method, how to represent user behavior published by [Chen, 2018] that is extending work of Vasile [Vasile et al., 2016] is based on methods that are well known in NLP. It is mentioned in Section 2.1.1 that some researchers already used NLP practices to represent the behavior. However, this one goes **more in-depth**. It is based on work published by Mikolov [Mikolov et al., 2013a] called Word2vec and then extended by Quoc et al. [Le and Mikolov, 2014], where they propose distributed representation

of words (documents) in a vector space. This method is based on neural networks and have wide range of applications as can be seen in [Sagar Arora,] or [Ma et al., 2016].

2.1.3 Markov chains

Markov models are often used for predictive modeling, and many researchers experiment with this approach to model and predict user behavior. Markov model is a stochastic model where next state depends only on the current state. This method is presented in [Lu et al., 2006] [Jonáš, 2017] and [Deshpande and Karypis, 2004] to predict navigation paths of the users. Problems of this method are addressed by Amrich in his work [Jonáš, 2017] , where Markov chain is used for clickstream prediction. It is quite challenging to use this approach for modeling user behavior because it can be very computational demanding due to the high sparsity of state space of the higher order Markov approximations.

2.2 Clickstream clustering

Markov Chains are not usually sufficient for capture of complex behaviors of the users. Unsupervised user's clickstream clustering seems to be the more promising way how to retrieve interesting information about user groups. There are approaches based on different types of clustering. First approach mentioned in [Wang et al., 2016] and [Wang et al., 2017] is employing hierarchical divisive clustering. This method is fairly successful, however, it assigns only one cluster to each user, which is not always true in real-world datasets. This issue is then addressed by Silahtaroglu [Silahtaroglu and Donertasli, 2015].

The downside of clustering is that output of unsupervised methods does not necessarily provide meaningful assignments for humans. There is usually need for some checking what is actually in the produced clusters and evaluate that information manually.

2.3 Semi-supervised approaches

Wang [Wang et al., 2016] mentions that semi-supervised models are used for Sybil detection in different online applications. This application can be generalized to anomaly behavior detection. Baseline for semi-supervised classification is **label propagation** which can be useful in case of user's behavior labeling, because it performs well in discovering community structures in the networks [Raghavan et al., 2007].

2.4 Conclusion

In this chapter we presented some recent works on topic of user behavior modeling, clustering and semi-supervised learning. There is often mentioned connection between behavior representation and NLP. Our proposed models will continue in this trend and we will try to take a use of existing and well-known methods from that field.

Chapter 3

Theoretical background

3.1 Bag of Words

Basic and the most straightforward approach for a document representation is done by constructing bag of words vector. It starts by building a vocabulary, which is the list of unique terms that appear in the source documents. BoW vector then represents a single document as a vector D, where the length of D is equal to the number of the terms in the vocabulary and D_t corresponds to the number of occurrences (frequency) of term t in the document.

A significant drawback of this approach is that all the vectors are usually very sparse, given the fact that not every document does contain all of the terms of the vocabulary. Another issue with BoW representation is that it does not preserve any information about term order, which can be important in various applications. Despite its shortcomings, it is still used quite often as a quick baseline solution.

3.2 Inverse Document Frequency

TF-IDF (stands for Term Frequency - Inverse Document Frequency) addresses an issue with term importance, which is not present in BoW method. It was introduced back in 1973 by Jones [Jones, 1973] and is still widely used. It is based on an idea that some terms (such as the, a or and in the English language) tend to repeat a lot across multiple documents and therefore carry less information about the actual document.

TF in the name stands for **term frequency** which we discussed in Section 3.1 - it is a number of occurrences of the term in a document. **IDF** stands for **inverse document frequency**, and its computation is shown in Equation (3.1):

$$idf(t) = log\frac{1 + n_d}{1 + df(d, t)} + 1 \tag{3.1}$$

where n_d is total documents count in the set and $df(d,t)$ is the frequency of term t in document d. Actual term frequency is then calculated as shown in Equation (3.2)

$$tfidf(t,d) = tf(t,d) * idf(t) \tag{3.2}$$

A vector of terms frequencies are then usually normalized using l2 normalization because it is not usually necessary to maintain the length of the vector and it has some interesting properties discussed in Section 3.5

This method has more focus on important features than pure BoW. However, it does not solve the problem with a missing context of the terms and also the problem of often very sparse vectors.

3.3 n-gram

To introduce some sense of word context in previously mentioned methods, it is possible to group pairs, triplets or even more words. This method of grouping words is called **n-gram**. In formal saying n-gram is the consecutive subsequence of length n of some sequence of tokens $w_1 \dots w_m$ [ngr, 2018]. It is often utilized by one of the text vectorization methods presented in Section 3.1 and Section 3.2.

A process of building n-grams is quite simple and straightforward. We have to split the document (usually by spaces) into tokens, and that creates a list of unigrams. It is now possible to move a sliding window of size n through the list of the unigrams and with every step we get a new n-gram. This type of n-grams is built on a word level. In NLP applications it is quite common to build n-grams on character level, which provides more granularity and allows to build the more general model of language. [Pedregosa et al., 2011]

3.4 Latent Semantic Analysis

The latent semantic analysis is technique often used in NLP and information retrieval (IR) to discover possible relations among documents. In principle, LSA is an application of singular-value decomposition (SVD) to reduce the dimensionality of terms matrix produced by **BoW** or **TF-IDF**.

The output of LSA has lower dimension than original terms matrix, and therefore data are more suitable for various tasks performed on documents (such as clustering or classification). The main drawback of LSA (respective SVD) it has high computational complexity, and it is not suitable for large data.

LSA was first proposed by Deerwester et al. [Scott et al.,] and it was recently applied to clickstream data by Weler [Weller, 2018] to analyze the behavior of users and detect hijacked accounts.

3.5 Similarity of vectors

One of the most widely used metrics to determine how similar two documents are is **cosine similarity**. It has an advantage over the Euclidean distance in documents related tasks because it compares vectors in terms their of directions and is not affected by vectors magnitudes. This method takes two vectors and produces a number in a range from -1 to 1. In documents similarity applications vectors are usually non-negative, so, therefore, one angle cannot be larger than 90°. For documents vectors 1 means, that documents vectors are parallel and therefore similar and 0 means that vectors are perpendicular therefore dissimilar. Cosine similarity is calculated as shown in Equation (3.3) [Manning et al., 2008].

$$similarity(u, v) = cos(\theta) = \frac{u * v}{\|u\|_2 * \|v\|_2} \tag{3.3}$$

The complement of cosine similarity is called **cosine distance**, and it is calculated as $1 - similarity(u, v)$.

In some applications, such as k-means clustering, it is preferable to use Euclidean distance; however, this metric is not much useful for tasks, where we care more about direction of the vectors than their euclidean distance. It is possible to use Euclidean distance instead of cosine distance as long as the vectors are l2 normalized.

3.6 Distributed representation

Method called **Word2vec** that is designed to produce distributed word representation in vector space proposed by Mikolov [Mikolov et al., 2013a] finds success in many NLP related tasks. Shortly after publication, many researchers found out, that this method has broad spectrum of applications (applications are mentioned in Chapter 2).

Word2vec model is two-layer neural network and it can be implemented in one of two possible architectures - either **CBOW** (Continuous bag-of-words) or **skip-gram**. Both of these architectures are shallow neural networks, and they produce distributed representation of words. However, each of them is approaching this task in a slightly different way. The idea behind this model is that words with similar meaning occur in similar contexts. Both architectures solve a task where there is context w_i of the word w_i presented to the network and the actual word w_i is being predicted. This is happening in the case of the Continuous bag-of-word (CBOW). In case of the skip-gram architecture, the task is reversed. That means word w_i is presented to the network on the input and the context is predicted on the output. Words are fed into the network encoded in the form of one-hot vector - that means that word w_i is represented as the vector of zeros of a length that is equal to the size of the vocabulary with 1 at the position i).

There are two main hyperparameters that need to be set, in order to produce solid word embedding. The first parameter is the **size of context window**, which determines how many words there will be before and after the predicted, respectively guessed word. Size of context window is usually set to value between 10 to 15 in NLP applications [Mikolov et al., 2013a]. Next parameter is dimensionality of the embedding. This value is representing a number of neurons in the hidden layer of the network.

There are some interesting properties of embeddings produced by word2vec discussed by Mikolov [Mikolov et al., 2013b]. Computed vectors seem to be able to capture very well synonyms and semantic relations among words. It is also possible to use vector arithmetic and compute new vectors which will be close to the analogical words (in this case close mean that cosine similarity is high). For instance, we can see that $W(king) - W(man) \approx W(queen) - W(woman)$ or $W(France) - W(Paris) \approx W(Russia) - W(Moscow)$. These properties can be useful in many applications outside the NLP applications, and as it is mentioned in Chapter 2 it is also widely used.

3.7 Document extension

Word vectors are interesting, however, in many applications, it would be useful to have a vector representation of the whole block of the text (paragraphs or documents). The most straightforward solution is to sum or average the word vectors that made up the original document. However this solution does not provide any better results than simple bag-of-words approach [Le and Mikolov, 2014].

While word2vec works on the word level, an approach called **paragraph2vec** leverages abilities of word embeddings to paragraph (document) level. In paragraph2vec extension, there are two concepts similar to **CBOW** and **skip-gram** mentioned in Section 3.6 called **PV-DM** (Paragraph Vector - Distributed Memory) and **PV-DBOW** (Paragraph Vector - Distributed Bag of Words).

First architecture PV-DM extends original word2vec idea by adding new input nodes to the network that represents document identifier, which is also encoded as a one-hot vector on the input. This works as memory and adds context to the word vectors. On the output, the actual word is predicted. **Distributed Memory** is similar to **CBOW** architecture from word2vec.

The second approach for computing distributed paragraph representation is called PV-DBOW. This architecture is similar to skip-gram discussed in Section 3.6. On the input the document identifier is presented and on the output randomly sampled words from the paragraph are being predicted. This method does not perform as well as PV-DM. However, the output of those models can be combined and then produce even better results [Le and Mikolov, 2014].

3.8 Clustering

The main goal of clustering is to create multiple groups of items that share similar properties in the unsupervised manner. Two different approaches to the clustering will be discussed in this section. The first approach is centroid-based clustering that is represented by **K-Means** algorithm, which is a special case of distribution based Expectation-Maximization (EM). The second approach mentioned in this section is connectivity based **hierarchical clustering**.

3.8.1 K-means

K-means is quite an old clustering algorithm proposed in 1979 by Hartigan et al.[Hartigan and Wong, 1979] It assigns each point from the original dataset to one of the k centroids, where k has to be chosen apriori. The algorithm starts by selecting k points as centroids and then runs in iterations. In each iteration every point is assigned to the nearest centroid and position of that centroid is then recalculated as a mean of points assigned to it. Algorithm ends after an arbitrary number of iterations or when all centroids converge and do not change.

Often mentioned drawback of k-means its sensitivity on initialization, where positions of centroids are selected. It can produce diametrically different and locally optimal results. However, it is possible to use the semi-supervised method of k-means and place initial centroids with some sense of the clustered data.

EM algorithm similarity

EM algorithm is a generalization of k-means algorithm. In EM there are two steps called **expectation step**, which is equivalent to assigning points to the clusters and **maximization step**, which is equal to recomputing centroids.

3.8.2 Hierarchical clustering

In hierarchical clustering, there are two main approaches. The first one is bottom-up approach called **agglomerative clustering** and the second one is top-down approach called **divisive clustering**. Both approaches are computationally complex, and there is a need for heuristics in both cases, to be able to use those methods on larger-than-small datasets.

Bottom-up

The first method of hierarchical clustering is called **agglomerative**. Agglomerative clustering begins with n clusters, where n is the number of items

being clusterized. The algorithm then repeatedly in every step merges two clusters that satisfies linkage criterion. There are several linkage criteria for linking two cluster A and B:

Single $\min\{distance(a,b) : a \in A, b \in B\}$

Complete $\max\{distance(a,b) : a \in A, b \in B\}$

Average $\frac{1}{|A||B|} \sum_{a \in A} \sum_{b \in B} distance(a,b)$

The algorithm terminates when there is just one macro cluster that contains all items. Time complexity of a naive implementation of agglomerative clustering is $\mathcal{O}(n^3)$. Therefore, it is not applicable to any larger datasets.

Divisive clustering

This algorithm starts with all items in one big cluster and then splits that cluster into smaller ones with some other flat clustering method. The algorithm ends, when there are n clusters.

3.9 Semi-supervised learning

On the way between unsupervised learning and supervised one lays the **semi-supervised learning**. Standard **SSL** tasks use a small amount of labeled data together with a large quantity of unlabeled ones to improve the learning accuracy. Acquiring labeled data is not always easy. However, semi-supervised methods can produce better results than unsupervised ones entirely without the labels and on the other hand supervised methods with just a small amount of labeled data. There is also another option, how to incorporate some specific knowledge into the unlabeled data - by using **constraints** [Chapelle et al., 2010]. To make SSL meaningful task, there are three assumptions, where at least one of them has to be fulfilled:

Smoothness assumption if two points x_1 and x_2 are close, their outputs y_1 and y_2 should be close as well

Cluster assumption if two points share the cluster, they likely share a label

Manifold assumption the high-dimensional data lie on a manifold

In this section approaches that make use of those assumptions to assign labels to unlabeled data will be discussed.

3.9.1 Label propagation

This is an iterative algorithm for semi-supervised learning based on the propagation of the labels on the graph. It assumes smoothness of the data, so if two input points are close, outputs should also be close. To describe the algorithm further lets presume that there are labeled pairs $(x_1, y_1) \dots (x_l, y_l)$, and unlabeled pairs $(x_l + 1, y_l + 1) \dots (x_u, l_u)$, and classes C. The goal is to estimate labels of unlabeled data based on our knowledge of labeled ones.

The algorithm starts by building a fully connected graph, where each edge has assigned weight w_ij, where w_ij is larger when the i and j are closer. Next let us define a probabilistic transition matrix T as follows:

$$T_ij = P(i \to j) = \frac{w_ij}{\sum_{k=1}^{l+u} w_kj} \tag{3.4}$$

T is the probability of propagation label from node i to node j. All nodes have a soft assignment of a label represented by a probability distribution over labels. Let us define the matrix of labels Y of size $l + u \times C$, where each row of Y is representing class assignment probabilities for observation x_i. The actual algorithm of label propagation is iterative, and in each iteration, there are three steps [Zhu and Ghahramani, 2002]:

1. **Propagate** - $Y \leftarrow TY$
2. **Normalize** - normalize rows of Y
3. **Clamping** - update known labels in Y

Iterations of the algorithm are referred as $t/$ Algorithm terminates after arbitrary number of iterations, where it may end up non-converged. Another option for algorithm termination is when the difference between two label assignment distributions Y^t and Y^{t-1} is below some arbitrary threshold [Pedregosa et al., 2011].

Chapter 4

Design and approach

In this chapter, we present the main idea behind user behavior embeddings that represent recorded clickstream together with other events that occurred during user's visits. With this stated, we also introduce new term **eventstream** to generalize clickstream that includes other types of events than clicks. We will also describe the raw data provided by Smartlook and process of collecting and prepocessing them into the form which would be suitable for building behavior embeddings. In the last part of this chapter we will discuss different clustering and semi-supervised approaches that will be used to group similar visitors in Chapter 6.

4.0.1 Entities in the system

Original data provided by Smartlook are represented by four entities - **visitor**, **session**, **record**, and **event** (show in Figure 4.1). The atomic unit of the dataset is called event (the example of such an event can be seen in Appendix A). It contains information about low-level browser events, either invoked by a user (such as clicks or typing), or by a browser (e.g. JavaScript runtime errors) or the events emitted by the application. Events are formed into records, which represent visit of a single page. On higher level, records are grouped into sessions, where sessions are separated by 15 minutes of visitor's inactivity. Every session belongs to a visitor, that is recognized by an identifier. This identifier is kept in visitors browser cookies, and therefore it is possible to identify visitor among multiple sessions in one browser. Another option how to identify the particular visitor is to receive its unique identifier from the actual application, that knows the identity of the visitor (for example when the visitor logs in, it is then uniquely identifiable).

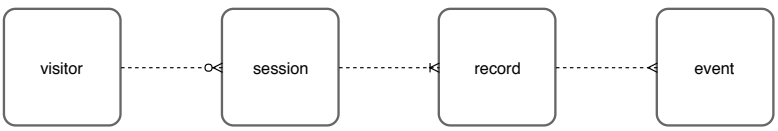


Figure 4.1: Relations of the original entities

Metadata fields of the events

Event entity as the atomic unit of the user's behavior record have some metadata fields stored with them. Not all of them are important for this work so that we will focus just on a subset.

At the most crucial field is called **eventType**. It represents several different types of possible events:

click is the most fundamental event, representing all clicks user performed in a browser. Clicks do not have any first-class semantic identification. Therefore there is a discussion in Section 4.2.3 about semantic clicks identifiers.

url represents every change of the state in the address bar. This is especially useful for SPA (Single Page Applications, where a change of the URL address does not always reload the whole page from the server.

text is emitted every time user's fill in some textual field (such as `<input>` or `<textarea>`). This event is usually preceded by a **click** event.

error event is triggered by a user's web browser when JavaScript runtime error occurs.

custom is programmatically emitted event from the recorded application. Its name is stored in field **eventName**

The rest of the fields are used to build features together with **eventType**:

pageUrl stores actual value of the address bar in time of the event occurrence.

value stores contextual data of the event (e.g. value of the text field or text of the error)

selector is CSS selector of the node in the DOM tree

elements stores list of more specific selectors in the DOM tree. Only the first element of that list is used in this work (represents actual element)

time represents milliseconds elapsed from the record start

props stores different types of information that does not have to be always present.

vid is visitor identifier

sid is session identifier

rid is record identifier

4.1 Proposed idea

In Chapter 2 we mention the striking similarity between textual data and click data. It seems that it is possible to transform clicks (resp. events) into sequences ordered by time of occurrence and treat them as strings. The proposed analogy between visitor and document is presented in Figure 4.2. Thanks to this observation, it is now possible to apply different techniques known from NLP to build embeddings of the eventstreams that represents user behavior.

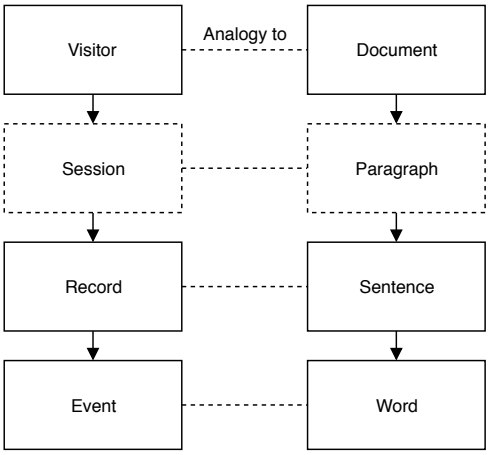


Figure 4.2: Analogy between documents and visitors

4.2 Preprocessing the data

Raw event stream data tend to be "noisy" and it needs to be preprocessed in order to use it further in modeling behavior. There are several issues addressed in preprocessing steps such as extracting the subset of pages for modeling, cleaning and clustering URLs and because clicks are not semantically identified, it is necessary to create identifiers for those events.

4.2.1 Extracting the set of the interest

It is not always desirable to analyze whole complex application, that has multiple different sub-applications. As an example: Facebook consists of multiple smaller applications, such as events, chat or advertisement manager and the main application serves more as a dashboard to access any of them. It makes sense to analyze just a subset of the application. Therefore the first step of the data preprocessing is to extract only events relevant to some part of the application. This is done by matching URLs with the regular expression and this approach assumes, that URLs in the application have some semantic meaning.

4.2.2 URL clustering

Each event has assigned URL address of the page of its occurrence. This is especially important for events of type `url` (respectively `navigation`) where there are as much unique `url` events as there are unique URL addresses. This approach is not applicable for larger websites, where URL address usually holds much information. We state an assumption that similar URLs represent similar behavior. After observation of common URL addresses, there can be seen the pattern, because URLs usually follow a directory-like structure or at least contain some information, which makes them good candidates for clustering.

At first, it is necessary to preprocess all URLs. Standardized scheme of the URI is following:

Listing 4.1: URI Scheme

```
scheme:[//authority]path[?query][#fragment]
```

Listing 4.2: URL query

```
key1=value1&key2=value2
```

The query is a string representing key-value assignments (Listing 4.2). We start with the strong assumption that everything except path and query can be stripped. A domain can be stripped, because we made an assumption that the model is only for one domain only. With stripping the domain, it is also possible to remove the `scheme:` and `[//authority]` because those are same in all cases and even if they are not, there is not much useful information about the behavior of the visitor. Last part of the URL - fragment - can also be removed, because it is usually used just for navigating in terms of one page.

Next step of preprocessing the URLs is tokenization. The URL is now in the form of `path[?query]` and therefore can be split by any of those characters - `/ ? &`. It is also quite common to use dashes (`-`) or underscores (`_`) in the URLs, so it seems to be reasonable to use those for splitting the address into the tokens.

To be able to clusterize the URLs, it is necessary to represent them in the vector space. For this case, L2 normalized TF-IDF vectors mentioned in Section 3.2 are the good-enough choice.

For clustering, hierarchical agglomerative algorithm is used, with use of cosine similarity and average linkage (mentioned in Section 3.5).

4.2.3 Click identification

In Appendix A is an example of a raw click event. It may seem it is uniquely identified by its selector or selector and URL combination. However, those

two values would generate the long and sparse feature vector. In many related works, every click event is semantically labeled, and it is possible to identify same click action between different parts of the website. In case of a provided dataset, no such information is available due to its generic nature. To address this issue, four different approaches how to generalize the click identification.

Text value

First and most straightforward approach how to identify a button or a link is by its caption. This approach would be probably fine for some simple websites, however, in larger applications, there could be some issues. One of those issues is using images instead of text, so it would be challenging to retrieve any semantic meaning of the actual content. Applications may also be translated into multiple languages, so one button would have a different label in some two language mutations and therefore different identifier. Last steps are about discretizing time component and building the sequences of events.

CSS selectors

In web browser, all the elements that are present on the web page are kept in a structure called a **DOM tree**. Every element in a DOM has its name and some assigned attributes. To identify particular element, there are several options that can be used to navigate around the tree and to identify the nodes by a variety of criteria. The first option is to use XPath, however, in case of this work, XPath queries are not available in the dataset. Other options are to use CSS selectors, that aim to solve the same problem as XPath in more expressive and shorter way.

CSS selector is a string where single subtree selectors can be separated by tree traversal characters - space, + sign or >sign. Example of CSS selector can be seen in Listing 4.3

Listing 4.3: Full CSS selector example

```
HTML BODY DIV.box BUTTON.blue.wide
```

The downside of this approach is that it does not generalize well and for example one button may have different selectors even in term of one page, when it is moved somewhere else in the DOM tree.

Reduced CSS selector

CSS selectors can represent the path to the same node in many different unambiguous ways. However, it does only apply in the context of a single page. Between two different web pages, where each of them has a different DOM tree, the selector from one tree does not necessarily represent the same

node in the second tree. To resolve this problem, we can make an assumption, that only the last part of the selector, which represents the element with its name and id attribute or class attribute, is sufficient to represent semantically similar among multiple pages. Example of this can be seen in Listing 4.4.

Listing 4.4: Full CSS selector example

```
BUTTON.blue.wide
```

Reduced CSS selector with context

The idea of reduced CSS selector is probably too wide to be practically used. One reduced selector may have different meaning among multiple pages. To present some context to this selector, we add a cluster label of the page it occurs on to the selector. This is based on assumption, that selector (e.g. `BUTTON.blue.wide`) has similar meaning on a landing page, but in the application settings, its meaning is probably different.

4.2.4 Discretizing time component

A dataset of the events contains time as a continuous component and in order to build a discrete sequence from the user's events we have two options, how to deal with the time between the events.

The first approach is to get rid of the time component between two events and preserve the only order of the events as they occurred in time. This method will not be able to capture time differences between two events and losing this information may produce unwanted results in an application where knowledge of time is essential. For example, two users can produce the same sequence of events such as *Visit login page* → *Type username* → *Type password* → *Click login button*. First of the two users logs in instantly, however, the other one hesitates and does not know where is the *Login button*. This can be considered as two different behavior profiles, and without knowledge of time elapsed between those events, it is not possible to discover the difference between the profiles.

To introduce the sense of time to the sequence, we add a **sleep** event between every pair of the events in the sequence. The process of building sequences with discrete time events is shown in Figure 4.4. For purposes of this thesis, we introduce for new events called *sleep events* shown in Figure 4.3.

We may notice that **sleep-10** in Figure 4.3 is half-open. This is because some types of events (usually emitted from the code) can occur in the almost same moment, and it does not have any practical meaning to capture intervals in higher resolution than 1 second. Too much granularity would introduce unnecessary noise into the data and would not provide any meaningful features.

sleep-10 $(0\ seconds, 10\ seconds]$

sleep-60 $(10\ seconds, 60\ seconds]$

sleep-600 $(60\ seconds, 600\ seconds]$

sleep-inf $(600\ seconds, \infty)$

Figure 4.3: Sleep events that are used to mimic time between two consequent events in the sequence

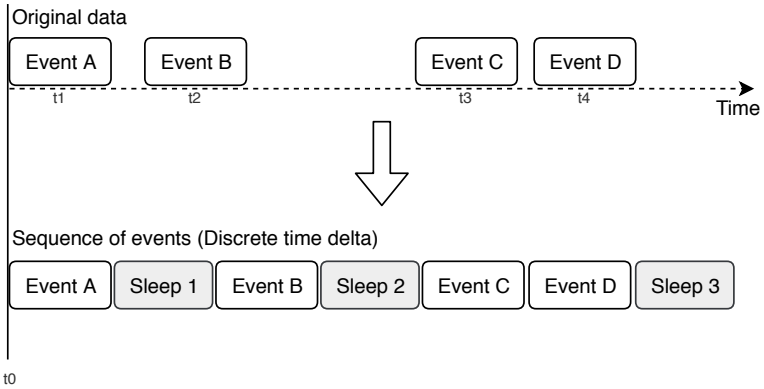


Figure 4.4: Adding time delta between events in sequence

4.2.5 Building sequences

After discretizing time component and preprocessing the data, we proceed to build **sequences of events**. This sequence contains string event descriptors Listing 4.5 of all events ordered by time of occurrence. This allows us to perform different operations on the sequences with time-locality quite efficiently. Event descriptor is shown in Listing 4.5. It consists of mandatory part *event type* and an optional numerical part *event identifier*. Those parts are separated by a dash. Event identifier represents the numerical identifier of a click (Section 4.2.3) or an URL (Section 4.2.2)

Listing 4.5: Format of event descriptor.

```
<event type>[-event identifier]
```

Sequence contraction

To reduce the amount of noise caused by repeated actions (e.g., repeated click on a scroll bar or "rage" clicking on a button) we introduce **sequence contraction**, that takes advantage of the time locality of the sequences. This process goes through the sequence and removes all consecutive duplicates, that share same event type, and the event identifier.

4.3 Behavior Embedding

In this thesis, we propose two different methods for building embeddings of users eventstreams. Both methods are using sequences presented in Section 4.2.5, however, each of them has a different approach how to find ideal representation.

The first approach use more traditional techniques well known from NLP. Behavior embeddings are based on n-grams and their frequencies. As an extension to this SVD is then used for dimensionality reduction (a technique known as Latent Semantic Analysis in NLP) to possibly improve clustering performance by lowering high dimensionality of computed vectors.

Second proposed approach to produce embeddings is based on relatively new work called paragraph2vec [Le and Mikolov, 2014]. This method has architecture very similar to word2vec by [Mikolov et al., 2013a], but it extends its capabilities to build embedding for whole documents. In its core, it is a shallow neural network that is able to learn embeddings of the documents. We use this to compute vectors from the user behavior sequences. To capture common consequent events, we also include an extension for phrase extraction to describe them.

4.3.1 Frequency Based Model (FB-k/d)

The first proposed approach is based on the assumption that the user behavior is a vector of weighted event frequencies. This model is referred as **FB-k/d**, where k represents value of n in n-grams (discussed in Section 4.3.1) and d is optional argument that represents application of SVD for dimensionality reduction of embedding to d. Those frequencies are weighted by idf (Inverse Document Frequency) to diminish the influence of events that occur too often and therefore do not provide much information about actual event sequence.

The process of building this model can be divided into four steps:

1. Build a set of possible features F and call it *vocabulary*
2. Filtering vocabulary
3. Build feature vectors
4. (optional) Perform dimensionality reduction using SVD

A downside of this approach is that it poorly generalizes and cannot recognize new events either sequences that were not in a vocabulary before.

Building a vocabulary

Baseline method takes every single event in each sequence without any context (unigram) as a feature. In some cases, the context may not be necessary,

and this method can produce a meaningful representation. However, without context, it cannot capture order of the events.

To preserve ordering on a local level, we introduce context by extension to n-grams. For each sequence, we build a list of all $1 \dots k$-grams, where k is some arbitrary number and uses all the produced n-grams as features. This approach can dramatically increase the number of features in the vocabulary. However, it captures much more information, and it can be scaled quite easily and predictably.

Filtering a vocabulary

To reduce the size of the vocabulary, we filter out those features, which do not meet desired criteria of *document frequency*. For every feature t in the vocabulary, we calculate many occurrences throughout all sequences $df(t)$ and remove all features where $df(t) < 5$. This number was determined empirically from different observations and finding, that in a reasonably large set of event sequences there are some events (such as misclicks), that do not have any deeper meaning.

Building a vectors

With filtered vocabulary, it is now possible to construct vector representation of event sequences. Every sequence S_i is encoded as vector D_i, where $D_ij = tfidf(F_j, S_i)$ (computation of $tfidf$ is described in Equation (3.2)). We may notice that the resulting vectors are usually very sparse. This is a common issue with BoW representation when each of the documents contains only a subset of all possible features in vocabulary.

The produced vectors can be now passed further to the clustering step. However, it is not usually suitable to have a high dimensional sparse matrix. Therefore we use SVD to find a representation in a lower dimension space.

4.3.2 Neural Network Based Model (NN-d)

Second proposed approach to building a behavior embeddings from the event-streams is referred to as **NN-d**, where d stands for the dimensionality of produced embeddings. It is based on the skip-gram architecture, and it is utilizing an extension of this approach called DBOW (Distributed Bag of Words), which was designed to generate paragraph embeddings that well capture semantic relationships among them [Le and Mikolov, 2014]. The pipeline of this model is following:

1. (optional) perform sequence contraction
2. (optional) build list of common n-grams in all sequences and replace frequent n-grams by a single event

3. discard events with less than 5 occurrences

4. build list of tags and sequences assigned to them

5. train model on pairs of tags and events

We start off by stating, that every event sequence has assigned an arbitrary number of *tags*. The tag is usually only one and contains the identifier of the user that produced that particular sequence. However, one tag can belong to more than one sequence (we will utilize this option later on). The architecture of the neural network is shallow, with only one hidden layer. The input of the network is one-hot encoded tag t_i that belongs to some event sequence. The next layer is hidden with d neurons, where d is dimensionality of the embeddings. The activation function of the hidden layer is linear. Finally, the output layer is one-hot encoded event j from the sequence, with softmax activation function. During the network training, algorithm iterates over all sequences, their tags, and events and trains the network on tag-event pairs.

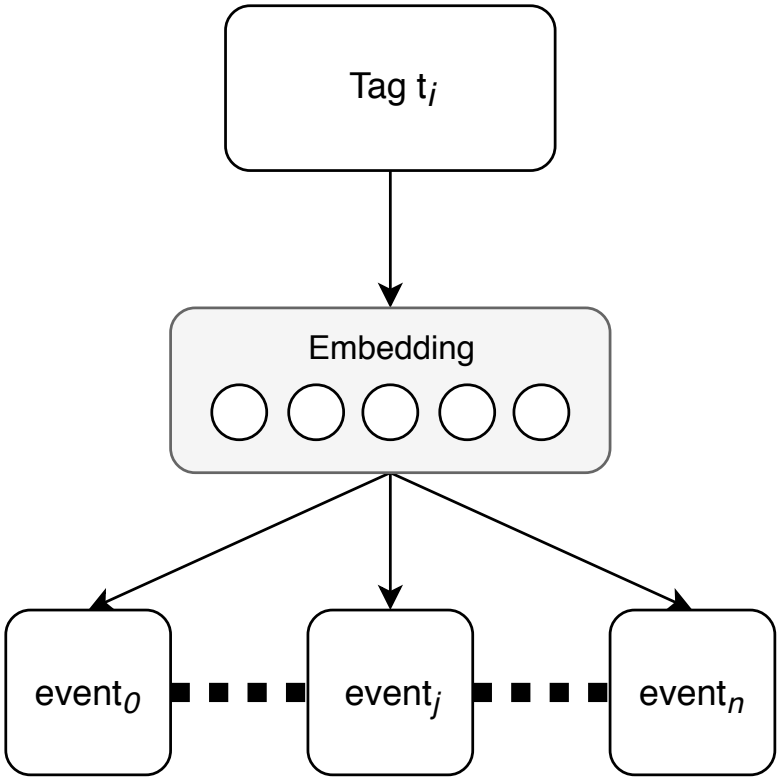


Figure 4.5: The architecture of the paragraph2vec network applied to the event sequences. On the input goes one-hot encoded tag t_i belonging to some sequence and is trained to predict one-hot encoded events from that sequence.

Common n-grams

Using DBOW model, as the name suggests, does not consider any context of the events. One possibility is (as proposed in the previous model) to add all possible n-grams to the dictionary and then train the model. However, as proposed in [Mikolov et al., 2013b] it is not necessary to add only those, that meet the desired criterion. In this thesis, we use scoring function to filter bigrams 4.1:

$$score(e_i, e_j) = \frac{count(e_i e_j) - min\_count}{count(e_i) * count(e_j)} \tag{4.1}$$

where e_i and e_j are events from the vocabulary and $e_i e_j$ means that e_i is directly followed by e_j.

To get longer sequences, we repeat this bigram building process one more time to get frequent trigrams. We also consider all *sleep events* as stop-words. That means those events are not taken into account and are skipped during the building of bigrams. For purposes of this work the $min_count = 5$ and score threshold, to add bigram to the vocabulary is set to 10.

Conclusion

The main advantage of this model is that it produces dense vectors with arbitrary dimension d. However, this approach has the same disadvantages as Section 4.3.1, and that is lack of generalization for unseen events. When the model observes an unknown event, it silently skips that one.

4.4 Clustering of embeddings

There are several options, how to cluster behavior embeddings produced by one of the proposed methods. In Section 3.8 we discussed well-known clustering algorithms and it is a task for experiments to determine which clustering algorithm suits best-given type of data. However, there are some parameters of the clustering that need to be addressed. In this section we start with a discussion about selecting the number of clusters, and it is without any prior knowledge about the nature of the clusters in the data quite complicated task. Next, we focus on selecting distance measure for the clustering algorithm and we end up the section by describing divisive clustering algorithm with feature pruning.

4.4.1 Selecting number of clusters

Selection of k in clustering algorithms is an uneasy task, especially when there is no prior knowledge about the cluster structure in the data. Some

clustering algorithms, such as DBSCAN do not require specification of the number of clusters, because it does reveal the clusters during its run. On the other hand, there are for example k-means or k-medeoids, that require k explicitly specified. There are many techniques that may help to decide about the number of the clusters. There is "rule of thumb" method saying that $k = \sqrt{\frac{n}{2}}$ where n is number of observations. However, this is not very exact so that we will take a look at two different methods - elbow method and silhouette.

Elbow method

First method that helps with detection of value k is so-called **elbow method**. This method is applicable only to centroid-based clustering algorithms. We start with a small number of clusters k, and we repeat following actions:

1. run K-means with k
2. calculate sum of squared errors SSE as following: $\sum_{i=1}^{k} \sum_{j \in C_i} \|x_j - \mu_i\|^2$
3. if the difference from last run is below a threshold, terminate. Otherwise increment k and go to step 1

To better understand elbow method, we may plot the values of SSE according to k and look for an elbow in a plot. Location of the bend is usually an indicator of an appropriate number of clusters. This method is dependent on the threshold, that determines the elbow.

Silhouette

This metric proposed by [Rousseeuw, 1987] provides an internal evaluation metric for determining quality of a clustering. Silhouette value is measure how similar is object i with other objects in the same cluster compared to objects in other clusters. Value $s(i)$ is calculated as follows:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \tag{4.2}$$

where $a(i)$ stands for the average distance between i and all the other points within its assigned cluster C_i. Now we define function $d(i, c)$, which equals to the average distance between i and all the points in the cluster $c \in C \setminus C_i$. Lastly, we define function $b(i)$:

$$b(i) = \min_{c \in C \setminus C_i} d(i, c) \tag{4.3}$$

Value of the silhouette is in the range from -1 to 1, where -1 means that the data are clustered wrongly and match better neighboring clusters. On

the other hand, values approaching 1 are the sign of good cluster assignment. If $s(i)$ equals to 0, then i is laying on the border of two clusters.

Computing just single silhouette value is not very useful, so there are two methods how to use this value. The first option is to compute the average of $s(i)$ for all data points i. This averaged value can be used to determine the number of clusters (goal is to maximize silhouette). Another option is to construct silhouette plot, where for each cluster values of all $s(i)$ are calculated and sorted per cluster. Then all the computed and sorted values are plotted as horizontal bars, and quality of the clustering can be checked visually.

4.4.2 Distance measure

To measure the distance between two points in clustering, we may choose different metrics. Most common distance metrics is Euclidean distance, that is calculated as follows:

$$d(a,b) = \sqrt{\sum_{j=1}^{k}(a_j - b_j)^2} \tag{4.4}$$

This distance metric is used for k-means clustering. To use it in context of this thesis, we always have to normalize input vectors as L2-norm. For agglomerative clustering algorithm, we use cosine distance which is described in Section 3.5.

4.5 Semi-supervised learning

This section will discuss two different approaches to semi-supervised learning applied in this work. The first approach is label propagation with k-nearest neighbors and the second one is based on training NN-d model with knowledge about some classes.

4.5.1 Label propagation

Method of semi-supervised learning presented in Section 3.9.1 appears to be suitable for this work, thanks to its tendency to form communities in the graphs. This property may be beneficial thanks to the assumption, that users with similar behaviors should be close to each other in the vector space.

To determine weight in the graph for label propagation, we relax the condition of full connectivity by building k-nearest neighbor graph. That means in case of label propagation weights, that $w_ij = 1$ when j is among k nearest neighbors of i. Otherwise the weight of the edge $w_ij = 0$ (that means there is no edge between two nodes in the graph). This approach highly

reduces memory requirements for further computations, because of the high sparsity of the graph. The fully connected graph would take $\mathcal{O}(n^2)$ memory, in comparison with $\mathcal{O}(k * n)$ where k is the number of neighbors and n is a number of labeled and unlabeled points.

4.5.2 NN-d

The second method of semi-supervised approach is based on training NN-d behavior model with some knowledge about labels. In Section 4.3.2 is mentioned that it is possible to train every sequence with one or more tags. We utilize this possibility and train NN-d model where every sequence have an identifier as a tag, and some of the sequences also have a label. Then during the training phase of the neural network, we present the sequence to the network either with an identifier or a label. This approach then allows us to get the representation of the label and then compare cosine distance between label and identifier embeddings. The classification of unseen labels then will be done by calculating all similarities between embeddings of labels and unlabeled sequences and assigning a label that is most similar to the sequence.

4.6 Summary of the chapter

This chapter presented approach how to transform visitor's raw data into sequences of events and then build embeddings of those sequences in the vector space. Those vectors can be clustered into groups of similar behaviors. The whole process is displayed in 4.6 and in the next chapter we will cover its implementation. and describe technical details of single steps.

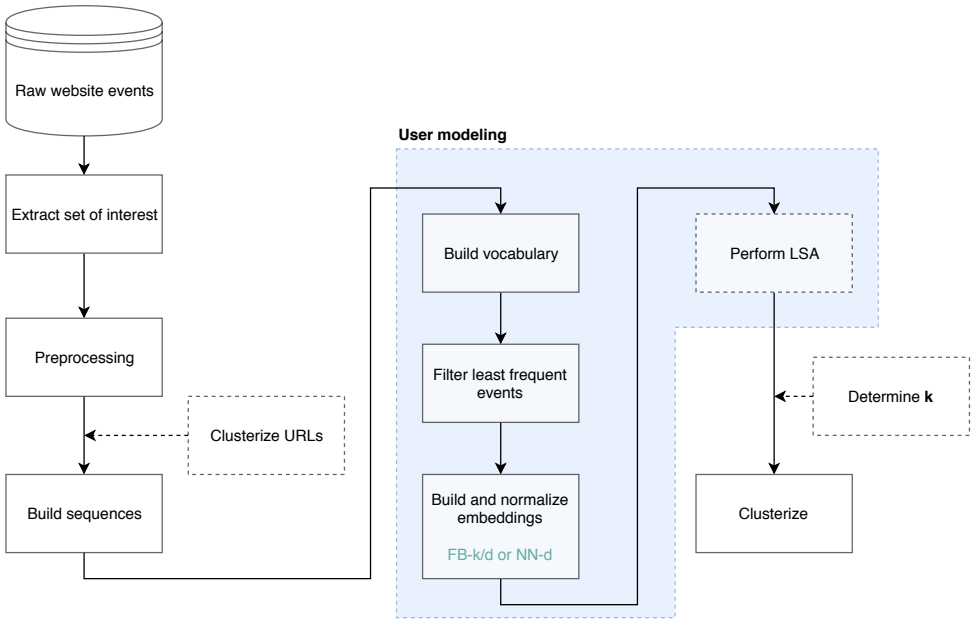


Figure 4.6: Architecture of the pipeline

Chapter 5

Implementation

5.1 Technology stack

For the implementation of the thesis, a variety of technologies is used. At first, the data retrieval from the ElasticSearch is implemented in **Node.js**. It is JavaScript based runtime environment, and it was fairly easy to integrate with existing Smartlook technology stack which is based on it. The actual data processing, models training, and evaluation is then implemented in Python 3.5 (build from the Anaconda distribution). All the data exploratory tasks and prototyping were done in Jupyter notebooks which is very flexible and interactive, so it is fairly easy to quickly explore multiple solutions.

For data retrieval, the official ElasticSearch JavaScript client is used for its simplicity and low overhead. A retrieved dataset is then stored into CSV (Comma Separated Values) files, which are easily portable. On Python side, there is wider range of the libraries used. The ecosystem of machine learning libraries is becoming very mature, and many implementations of popular algorithms is available there. Used Python stack stars with Pandas for data manipulation, NumPy [Jones et al., 01] and SciPy [Jones et al., 01] packages provided numerical routines and effective implementations for sparse matrices, scikit-learn [Pedregosa et al., 2011] and Gensim [Řehůřek and Sojka, 2010] for feature engineering and model building and finally matplotlib and MulticoreTSNE for the visualisation. Gensim library is implemented to be very effective in *cython* and in combination with linear algebra low-level library *BLAS* it delivers 72 times higher performance in comparison with baseline implementation using pure NumPy [Řehůřek, 2013]. All the used libraries are provided as open-source.

Data were obtained from the hosted Smartlook's ElasticSearch cluster on AWS. All the data processing and computations were done on a local machine running Mac OS with 2,7GHz Intel Core i5 and 16GB of RAM.

5.2 Dataset retrieval

Dataset was stored on AWS on the ElasticSearch cluster. Size of the raw dataset was about 15 million entries of various types and limitation of AWS is that the connection to the load balancer times out after 30 seconds. To avoid timeouts we implemented streaming wrapper around ElasticSearch scroll API, that allowed us to retrieve data in chunks that were transferable in 30 seconds time window.

To reduce dataset size beforehand, we used ElasticSearch DSL queries to extract the set of interest described in Section 4.2 by applying wildcard query on pageUrl field.

5.3 Behavior Embeddings

The frequency-based model is built on top of the magnificent scikit-learn library. To create baseline and FB-k/d model we use Tf-idf vectorizer and Truncated SVD that plays nicely together.

The NN-d model was implemented using well known Gensim library and its doc2vec model. This library provides a variety of NLP algorithms, and utilities and it is well optimized to work on CPU. This made this library good-enough for prototyping. However, training complex models with large corpora can be cumbersome due to its lack of ability to use GPU for training. First experiments with NN-d model did not make use of BLAS together with Gensim cython implementation and training of the model took about an hour depending on the size of the corpora and dimensionality of the embeddings. After adding support for numpy with BLAS, training time was reduced to minutes.

Chapter 6

Experiemntal part: Dataset and models

In this chapter, we cover preprocessing of the dataset and select ideal parameters for building sequences. Then we define a baseline behavior model and evaluate it together with other proposed models for building behavior embeddings.

The dataset provided by Smartlook contained raw data described in Section 4.0.1. Events were gathered from January to March 2018 and contain records of all users that visited some page on domain smartlook.com. To put dataset in light, Smartlook.com is a SaaS company that records the activity of visitors on partner websites and playbacks records in screen-capture-like fashion. Due to the highly private nature of the dataset, it is not possible to present its entries publicly. The original dataset contained around 10 million events, and this number was reduced to 300k after preprocessing, cutting of all users with sessions shorter than 30 events and extraction of the set of interest (described in Section 4.2). We excluded the actual dashboard of the Smartlook application (that means all URLs that contain `/app/`) from the set of interest, because of its high complexity and its redesign in February (therefore the events meta-data would not be consistent between two different designs). In the experiments, the dataset will be referred as SL300.

6.1 Dataset exploration

Dataset SL300 contained 301k events of various types (listed in Section 4.0.1) and represented 4419 different visitors. The number of events per visitor is displayed in Figure 6.1. There were 5803 unique URLs from different language mutations of the application. Those addresses were divided into 8 clusters, where one contained mix of very short URLs and others could be represented as:

1. player for shared records
2. platform documentation

3. application help
4. shared heatmaps of clicks
5. pricing
6. blog
7. contact

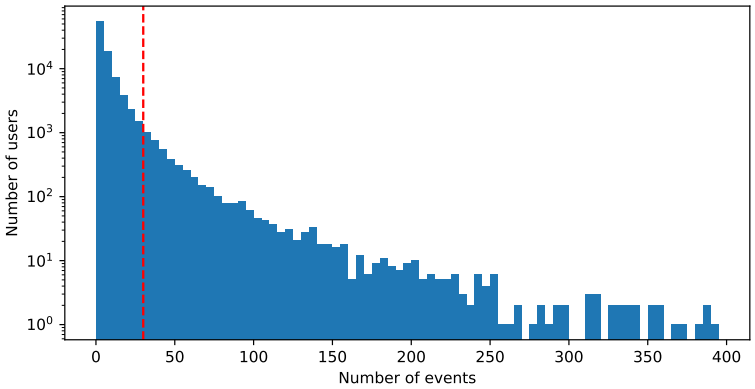


Figure 6.1: Number of events per user in the dataset with the reduced set of interest. A dashed red line represents cut-off for users with less than 30 events.

6.2 Baseline model

To have some point of reference for other models, we now define a baseline model. As a baseline implementation, we used the binary encoded vector of events. That means there was 1 when the event was present in the user's sequence. Otherwise, 0 was set. This model can be thought of as sum of one-hot encoded feature vectors with values clamped in interval $\langle 0; 1 \rangle$.

6.3 Cross-validation

All evaluations of the supervised tasks were performed using k-fold cross validation, where $k = 5$. This method splits data randomly into k groups, then the model is trained on $k - 1$ groups and the remaining group is used for validation. This process is repeated k times and each group is used exactly once for validation. Results of all k runs are then averaged.

6.4 Building a sequence

After cleaning the data, we had to build a sequence. For the further experiments, we evaluated multiple options in encoding behavior into the sequence

and selected the best performing one. Evaluation presented in Table 6.1 was done by performing the supervised task. As a classifier we used logistic regression evaluated with 5-fold cross-validation. As a training dataset we used embeddings from baseline model for every examined sequence and as a label for the users' models we used knowledge about their state - whether the user is logged or anonymous.

Parameters for building sequences were following (further explanation can be found in Section 4.2.5):

SL300-A used complete CSS selectors and sequences contained just a click events

SL300-B clicks and inputs were identified by their values and url events were identified by the number of a cluster

SL300-C clicks and inputs were identified by reduced CSS selectors and url were not clustered

SL300-D clicks and inputs were identified by reduced CSS selectors combined with URL clusters and url events were identified by the number of a cluster

SL300-E clicks and inputs were identified by reduced CSS selectors combined with URL clusters and url events were not clustered

Based on results presented in Table 6.1 **SL300-C** will be used in all further experiments.

	Accuracy
SL300-A	0.8933 ± 0.0286
SL300-B	0.8866 ± 0.0219
SL300-C	0.9050 ± 0.0163
SL300-D	0.9011 ± 0.0261
SL300-E	0.9036 ± 0.0156

Table 6.1: Accuracy of Logistic regression on different sequence types

6.5 Behavior embeddings

In this section, we evaluate the quality of the embeddings that were produced by both proposed methods. For evaluation of the quality of the embedding we use the same supervised task as was used in Section 6.4.

6.5.1 Frequency Based Model (FB-k/d)

Frequency-based model is implemented as described in Section 4.3.1. Hyperparameter of this model is a range of n for which a vocabulary of n-grams is constructed. Results of evaluation of different FB-k models are shown in Table 6.2. Highest accuracy was delivered by 1-5-grams model, which encapsulates all the other models together. The downside of this model is the size of produced embedding 32210. FB-k produces very sparse vectors and having the vector of this length may be inconvenient in many applications. Therefore we examined model FB-k/d which use SVD (singular value decomposition) to reduce the dimensionality of embeddings produced by FB-k. We experimented with $d = 80$ and $d = 300$ and results are presented in Table 6.3.

k	Embedding size	Accuracy
Unigrams	1222	0.8755 ± 0.0251
Trigrams	9197	0.8662 ± 0.0304
5-grams	7200	0.8282 ± 0.0418
1-5-grams	32210	0.8762 ± 0.0253

Table 6.2: Sizes of the embeddings and accuracy of FB-k models with different k

$d = 80$	Explained variance	Accuracy
Unigram	70%	0.8755 ± 0.0215
Trigram	35%	0.8726 ± 0.0300
5-gram	27%	0.8314 ± 0.0364
1-5-gram	40%	0.8714 ± 0.0228
$d = 300$	Explained variance	Accuracy
Unigram	90%	0.8755 ± 0.0214
Trigram	54%	0.8692 ± 0.0298
5-gram	47%	0.8414 ± 0.0349
1-5-gram	60%	0.8737 ± 0.0137

Table 6.3: Accuracy of FB-k/80 and FB-k/300 (dimensionality of embeddings reduced to 80 and 300, respectively) models with different k

6.5.2 Neural Network Based Model (NN-d)

Second proposed model based on PV-DBOW architecture has only one hyperparameter to set, and that is dimensionality of produced embedding (in

case of the PV-DBOW it is the number of neurons in hidden layer). Other hyperparameters of the model were fixed. The negative sampling was set to 5, and all models were trained for 30 epochs. All models were evaluated in the same way as a baseline model in Section 6.4. For all experiments, we build a vocabulary of events with frequent bigrams and trigrams and removed consequent duplicates in sessions. Results for various values of d are shown in table Table 6.4.

d	Accuracy
10	0.8844 ± 0.0098
80	0.8871 ± 0.0147
150	0.8884 ± 0.0060
300	0.8914 ± 0.0112
400	0.8875 ± 0.0108
600	0.8905 ± 0.0120

Table 6.4: Accuracy of NN-d for various values of d

6.5.3 Discussion of results

The results above show that model **NN-d** outperformed behavior model **FB-k/d** in all its configurations. However, none of the proposed models were able to surpass results of baseline model in the classification task.

Chapter 7

Experiemntal part: Clustering

In the next part of the experiments, we focus on clustering of the embeddings and evaluation of the clusters. We start with determining the number of clusters and visualization of the clusters. In the next part of the chapter, we explore clusters a describe users in them manually.

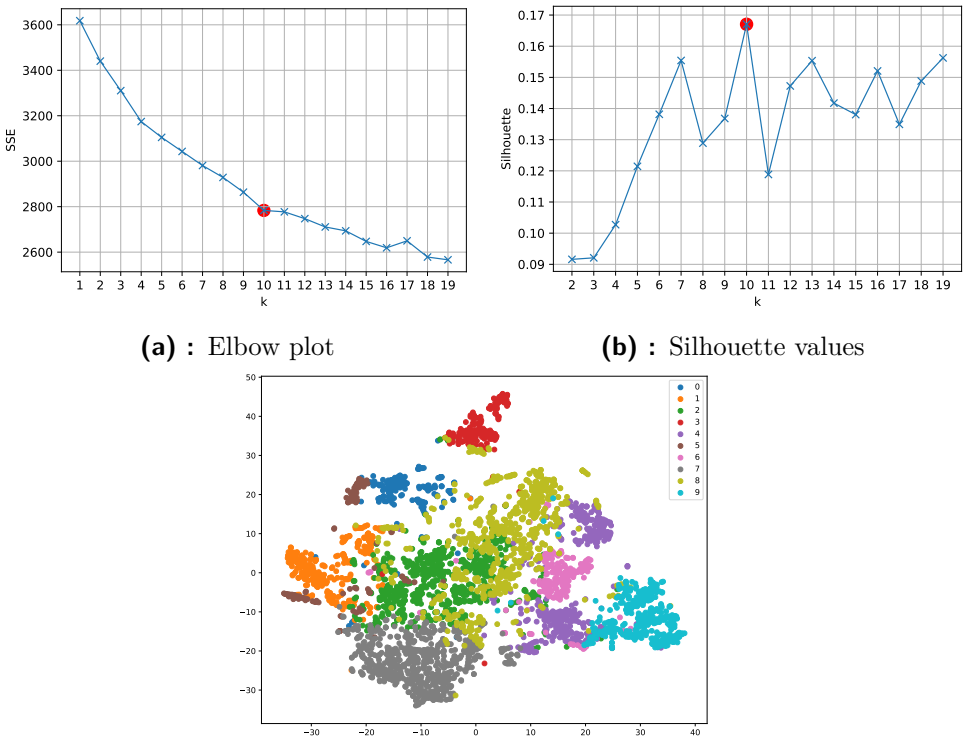
7.1 Visualization of the data

To be able to visualize high dimensional data in a 2-dimensional plot it is necessary to reduce a dimension of that space. Methods such as PCA or SVD does not always produce results, which are very informative when displayed in 2 or 3-dimensional space.

Method called **t-Distributed Stochastic Neighbor Embedding** seems to be much more suitable to visualize the data when PCA and another techniques fail. It is a nonlinear dimension reduction method that models points that were similar in the original high dimensional space near to each other in lower-dimensional space[van der Maaten and Hinton, 2008]. This technique is very popular to visualize data. However, interpretation of the produced results must be made carefully. Low-dimensional projection can sometimes show properties, that may not be true in the original high-dimensional space, thanks to the nonlinear nature of the projection.

7.2 Frequency Based Model (FB-k/d)

To present how the number of clusters was chosen, we decided to use a FB-1 model. We start with elbow plot Figure 7.1a. There is a visible elbow for $k = 10$. Therefore it is the candidate for the number of clusters. Then we move forward to the plot of silhouette values Figure 7.1b. There is a maximum also for $k = 10$, so therefore we assume that there are 10 clusters. To show the clusters in two-dimensional scatter plot, we used TSNE projection Figure 7.1c.



(a) : Elbow plot

(b) : Silhouette values

(c) : TSNE projection

Figure 7.1: Elbow plot and silhouette values for FB-1 model, TSNE projection, k-means, red dot indicates elbow/maximum

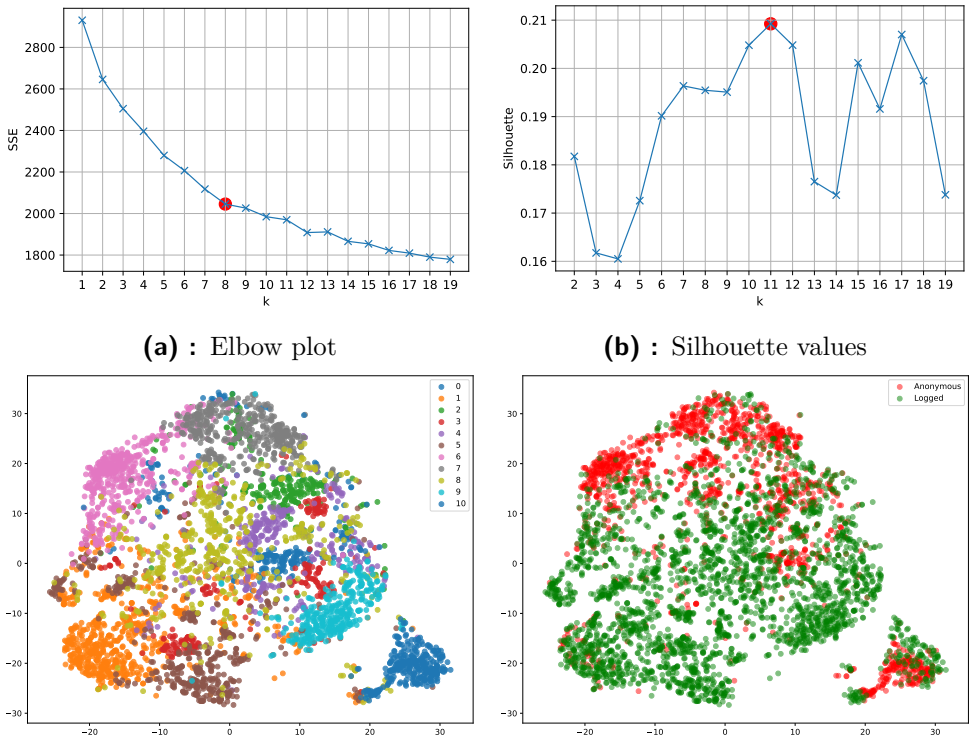
The next example is attached in Appendix C. It shows elbow plot and silhouette values for model FB-1-5/300. Elbow occurs for $k = 11$, however, silhouette value has its maximum for $k = 18$. Silhouette value for $k = 1$ is local maximum, therefore we may assume, that $k = 11$. This result is very similar to results from FB-1 model, therefore $k \approx 11$.

7.3 Neural Network Based Model (NN-d)

In this section, we examine clustering results for behavior embeddings produced by NN-300. This model had the best performance in supervised task presented in Chapter 6 therefore we chose that model to evaluate clustering. The elbow plot Figure 7.2a shows, that elbow appears for $k = 8$. The silhouette value has its peak for $k = 11$. The number of clusters may lay somewhere in-between those values.

More interesting observations can be done by taking a look at Figure 7.2d in comparison with cluster assignments in Figure 7.2c. Clusters 2, 6, 7 and 10 resemble clusters formed by anonymous users. With different types of labels (e.g., users with identified malicious behavior) the resemblance might be even more significant.

In Appendix D are shown results of agglomerative clustering. The distance metric was *cosine* and linkage was set to *average*. The results of this experiment were disappointing. Average silhouette value was much lower than in case of the k-means. This is a sign of poorly constructed clusters, and objects within cluster tend to be dissimilar to each other.



(a) : Elbow plot

(b) : Silhouette values

(c) : TSNE projection

(d) : Users with known labels

Figure 7.2: Elbow plot and silhouette values for NN-300 model, TSNE projection, k-means, red dot indicates elbow/maximum

7.4 Evaluation by hand

Evaluation of cluster analysis is not usually straightforward due to its unsupervised nature, and it often has to be evaluated by a human expert.

In this section we inspected results of clustering, to see whether the clusters contain any humanly interpretable assignments. To do an expert evaluation, we chose FB-1-5 and NN-80, both clustered with k-means, where k was set to 11 and 12, respectively.

7.4.1 Case study: Description of FB-1-5 anomalies (SL300)

This experiment was performed on SL300 dataset, and the behavior of the users was modeled using FB-1-5. To describe cluster anomalies we took a

look at the users most distant from the clusters centers.

To describe anomalies, we looked at the list of 3 users, whose average similarity to all cluster was lowest. The most outlying user had an average similarity of 0.012, and his behavior was just a set of rapid clicks on *Reviews* button. The second user with cosine similarity of 0.014 got stuck on one page and repeatedly clicked on an *emergency* button. Last examined user with an average similarity of 0.017 could not approve a form and continue further to the application.

7.4.2 Case study: Description of NN-80 clusters (SL300)

This experiment was done on dataset SL300 and behavior of user was modeled using NN-80. To evaluate this clustering, we find users that are closest to the centroids of the cluster. Then we transform event sequences to the readable form of CSS selectors and URLs and try to discover any human-readable meaning. Then we compare our findings with a visual representation of recordings in the Smartlook dashboard.

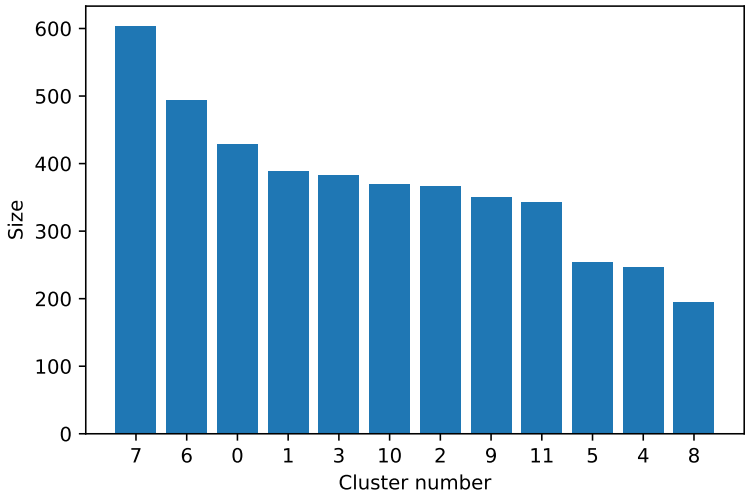


Figure 7.3: Sizes of the 12 cluster for NN-80.

After further inspection of all clusters and consulting inspection results with visual recordings, we can describe them as following:

0. anonymous visitors interested in pricing, features
1. users of Smartlook.com, that are just checking the dashboard (login through the form)
2. users of Smartlook.com, that are just checking the dashboard (persisted session)
3. visitors interested in pricing

4. no interpretable patterns
5. blog readers
6. visitors interested in features and partners of Smartlook
7. similar to cluster number 1, however, visitors are more active
8. visitors interested in awards and partners
9. visitors playing shared records (activity from a different part of the application than the rest)
10. users, that upgraded package after login

7.4.3 Interesting features

There were some interesting observations done while conducting various experiments. At first inspection of the users near to the clusters centers was not as informative as inspection of users that were most dissimilar from the centroids. Thanks to the observation of those users, we were able to identify misbehaving users of the Smartlook platform, which may be in practice even more valuable that inspection of familiar patterns.

The second experiment was done on embeddings of single events. This experiment showed interesting results in terms of learning semantic meaning of the events. In Section 3.6 we mentioned how word vectors can be added or subtracted and resulting vector will resemble semantically related word. Embeddings of single events shows similar properties (visualisation is in Appendix E). It was possible to perform an equation like a $click_{362} - click_{365} + click_{1236} \approx click_{363}$, where $click_{362}$ was login input, $click_{365}$ was just some input somewhere in the application and $click_{1236}$ was button, that submitted form of $click_{365}$. $click_{363}$ was suprisingly submit button of the login form. That means that event embeddings may carry a lot of information about structure of the application. This property can be used to reveal unknown usage patterns or for example detect which button caused an error, by exploring events nearest to the error.

Chapter 8

Experiemntal part: Semi-supervised learning

In this chapter, we discuss results of semi-supervised methods. First of two examined approaches is a label propagation. In conducted experiments, we assumed that users tend to form communities and therefore label propagation as described in Section 3.9.1 could be ideal for this kind of task. The second approach is based on NN-d model, where we could assign multiple tags to a single user and train the model with some of the tags and then by using cosine distance between predicted tags and users decide what label to assign to the user. To train semi-supervised models we used a small subset of labels used in Section 1.1.1 (labels were representing the state of the user - whether is he anonymous or logged).

8.1 Label propagation

In label propagation with k-nearest neighbors the main hyperparameter to tune is a number of neighbors, that will be affecting the value of a label. To show the differences between various models in the semi-supervised task of label propagation, we conducted an experiment, where we took labeled data from Section 6.4 and preserved 5% of labels. Then we trained label propagation model with different values of k, and then we plotted the accuracy of the predictions made by the model on a training set (rest of the dataset, that was not labeled). Results can be seen in Figure 8.1.

For conducted experiments, it turned out that NN-300 model performed best for all numbers of neighbors, although with some neighbors growing, all results became very close. The accuracy for two neighbors was ≈ 0.4, and that was not enough to provide any quality model for provided data. NN-300 model was trained with frequent bigrams and trigrams and removed consequent duplicates in the sequences. The other model FB-1-5/300 was trained with 1-5-grams, and then the dimensionality of the embeddings was reduced to 300.

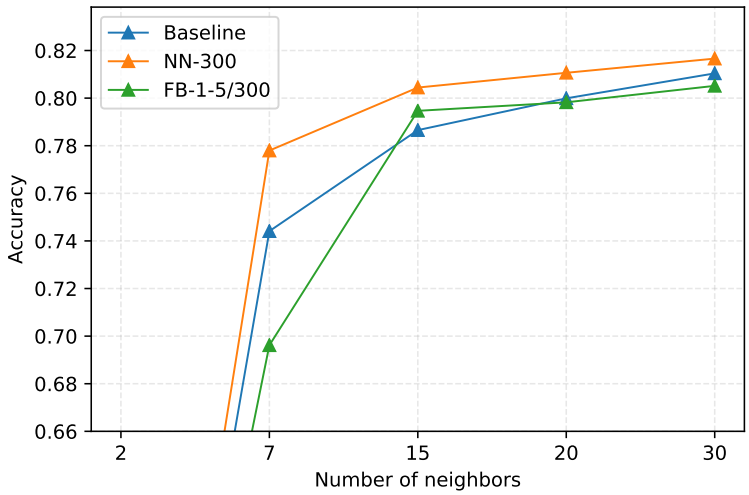


Figure 8.1: Influence of number of neighbors to accuracy measure on the test set (5% of labels used)

Results showed that NN-d model could capture well semantical similarity of users behavior and it is possible to group them well into communities.

8.2 NN-d

The second semi-supervised experiment is based on NN-d model. The unlabeled sequences are trained, as usual, that means identifier of the sequence is trained to predict the probability of words in the document and the labeled sequences are used for training twice - once with the identifier and second time with the label. Therefore, in the end, there is calculated embedding for every sequence and every label. Then we decide based on the cosine similarity between labels and sequences, what label to assign (assigned is the label that is the most similar).

To examine this method, we conducted an experiment where we tried to train different models with a various number of known labels, and then we plotted accuracy of the model on the test set. Results of the experiments with a comparison to baseline are shown in Figure 8.2.

Training NN-d model with labeled data may be combined with label propagation in a way, that we train a NN-d model with knowledge about labels and then we use embeddings from this model to build label propagation model, that uses same labels as NN-d. This should help tighten the communities with similar labels. Results of this are visible in Figure 8.2, where there is a red line, which represents label propagation model trained solely on embeddings produced by NN-300 and labels were used only in label propagation. The purple line represents results from label propagation model, that was trained on embedding produced by NN-300 with knowledge about labels. Both

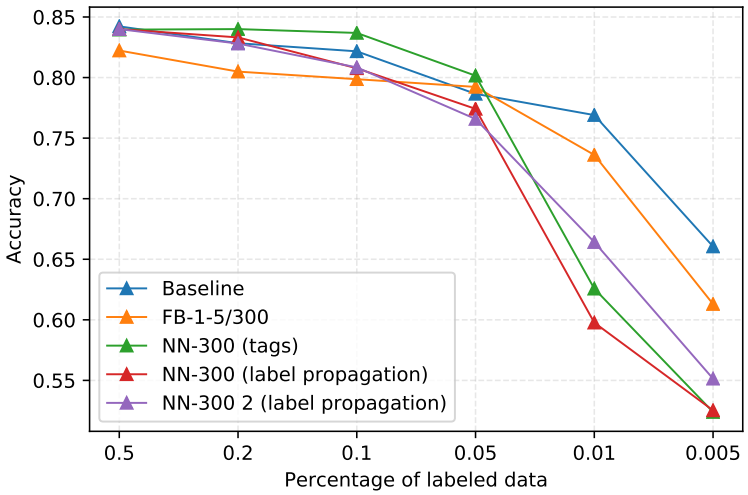


Figure 8.2: Influence of number of known labels to accuracy measure on the test set (all models were label propagation with 15 neighbors, with exception of NN-300 (tags)

methods performed fairly similar, and the only difference was when there was less labeled instances, the second model using embeddings trained with labels performed slightly better. Overall the best results were delivered by the semi-supervised solution NN-300 (tags), especially when the amount of labeled data was higher. However, accuracy declined rapidly with fewer data. With least amount of data, the best performing model was baseline behavior model with label propagation. This might be because of the binary representation, that can provide good results when there are only two labels that can be easily separated by few significant features.

Chapter 9

Conclusion

The goal of the thesis was to clusterize users based on their behavior. To do so, we used models build on a clickstream data and other events from the browser. In the first part of the thesis, we review current state-of-the-art methods for building behavior models together with techniques that are related to this problem. As it turned out, modeling of user behavior often internally resembles NLP tasks, so we took advantage of this knowledge, to explore a wider range of possible techniques. We also researched different methods of clustering and semi-supervised learning to find a suitable approach for the domain of our data.

In the next part, we then propose different methods for building behavior embeddings. The first method is based on weighted frequencies of the browser events with various sizes of the context window. The second approach is based on the technique called distributed bag-of-words. Embeddings produced by those methods were then used in other tasks in order to find clusters of similar users and train semi-supervised models.

Last part of the thesis is devoted to the experiments. Results were heavily dependent on a type of the task, that was performed. In supervised learning tasks, none of the proposed behavior models was able to surpass results of the baseline model. However, in semi-supervised tasks and clustering, the results of the second method were ahead of the others. In clustering task, best results were obtained by traditional algorithms, and among semi-supervised methods, highest accuracy was obtained by utilizing neural network approach.

Performance of produced behavior embeddings during different tasks is not significantly better than the baseline solution. However distributed bag-of-word captures semantic relations among the users very well, and when we applied this method on the level of single events, it produced some impressive results that would worth further research.

9.1 Future work

In this thesis, we presented baseline for using frequency vectors and distributed bag-of-words to represent user behavior, and we evaluated produced embeddings on various tasks. The results seem to be promising, and we propose three different options, how to extend this work. First of all, during the clustering experiments, exploration of the clusters was not easy. It would make sense to have some dynamic visualization of the clusters, maybe even integration with the actual web application, to better understand the context of the behaviors. Next possible direction could be deeper integration of word embeddings into the behavior models. The behavior of the users is often driven by the content of the web itself, so it would make sense to move focus from meta-information, such as CSS selectors and move to the actual content during modeling behavior of the user. Lastly, it would be interesting to explore more profound possibilities of detection of malicious behaviors by using distributed representations.

Appendix A

Raw recorded event


```
{
  "type": "event",
  "eventType": "click",
  "eventName": null,
  "sourceType": "website",
  "pid": "5ab0dfdd4faa3243480306a1",
  "vid": "aidqt6_65a",
  "sid": "ni6-gP40JcB",
  "rid": "amdkR2d01EY",
  "time": 554342,
  "timeCreate": "2018-03-28T14:47:38.183Z",
  "value": "Products",
  "pageUrl": "https://example.com/section/first-product",
  "selector": "HTML BODY DIV.row.text-center DIV.col-12",
  "props": [
    {
      "type": "internal",
      "name": "pageTitle",
      "value": "Example Product"
    }
  ],
  "elements": [
    {
      "classes": [
        "col-12"
      ],
      "tagName": "DIV",
      "nthOfType": 1,
      "nthChild": 1
    },
    ...
  ]
}
```

Appendix B

NN80 with 12 clusters

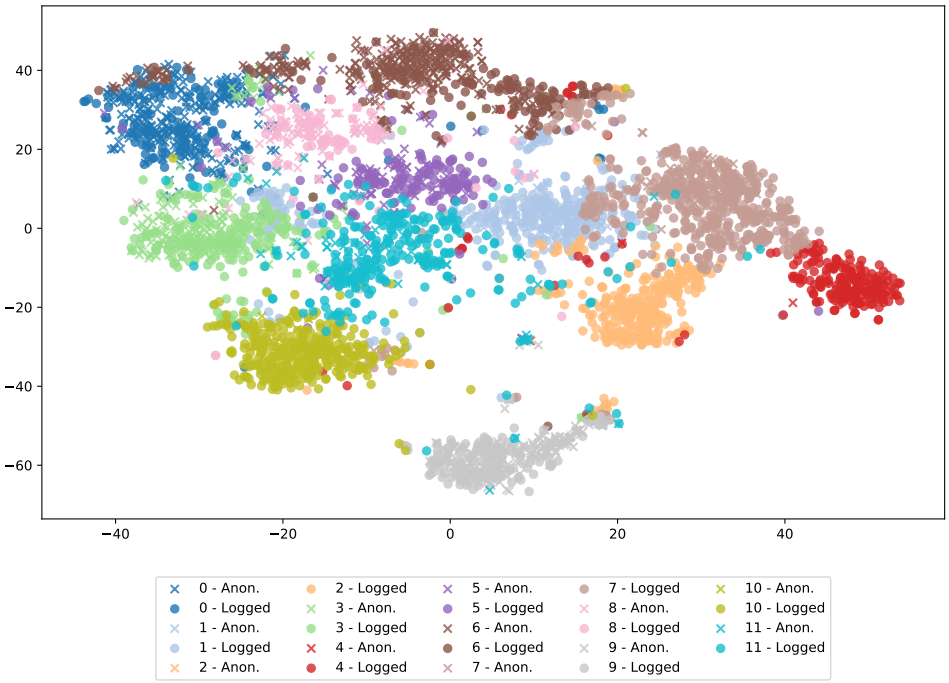


Figure B.1: Colored clusters of users in SL300 modeled by NN80 split into 12 clusters with K-Means. Each point represent one user and shape of the point resembles the state of the user.

Appendix C

FB-1-5/300 - Clustering

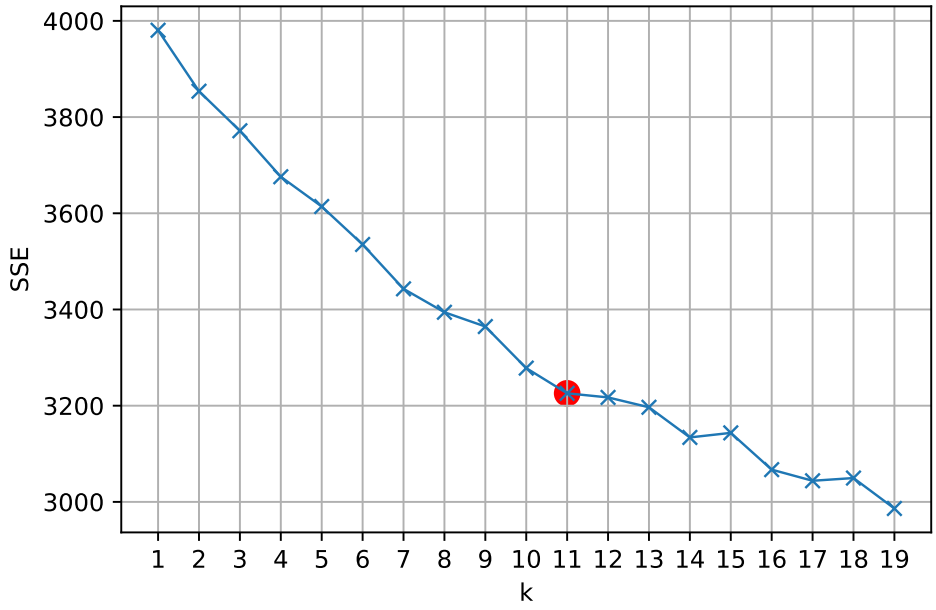


Figure C.1: Elbow plot

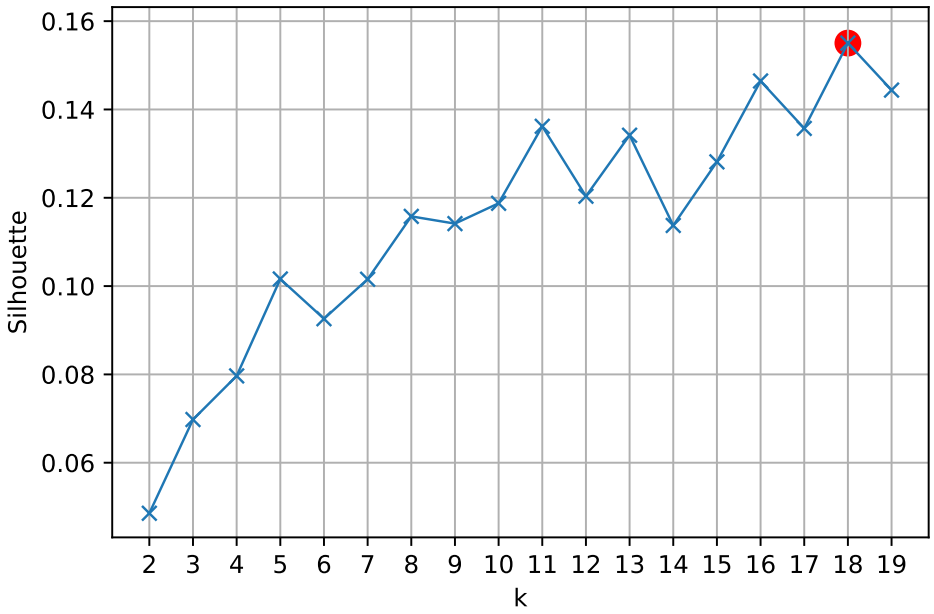


Figure C.2: Silhouette values

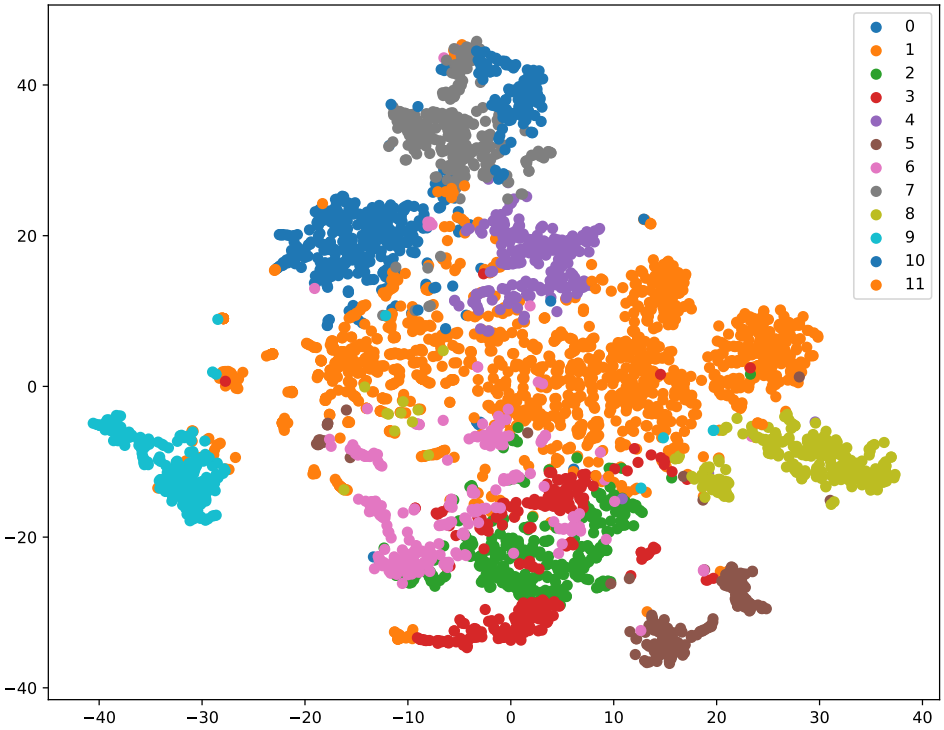


Figure C.3: TSNE projection

Appendix D

NN-300 - Agglomerative clustering

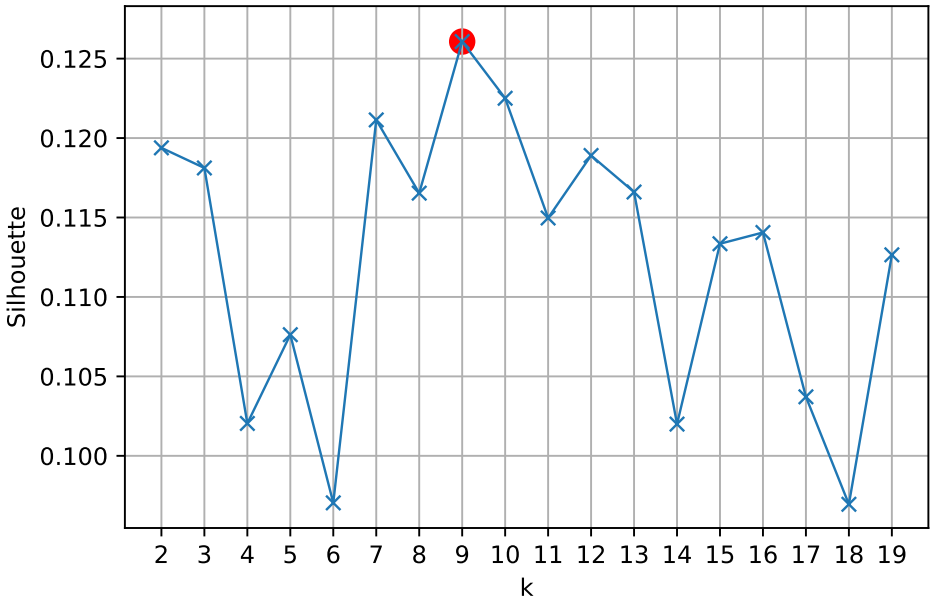


Figure D.1: Average silhouette values

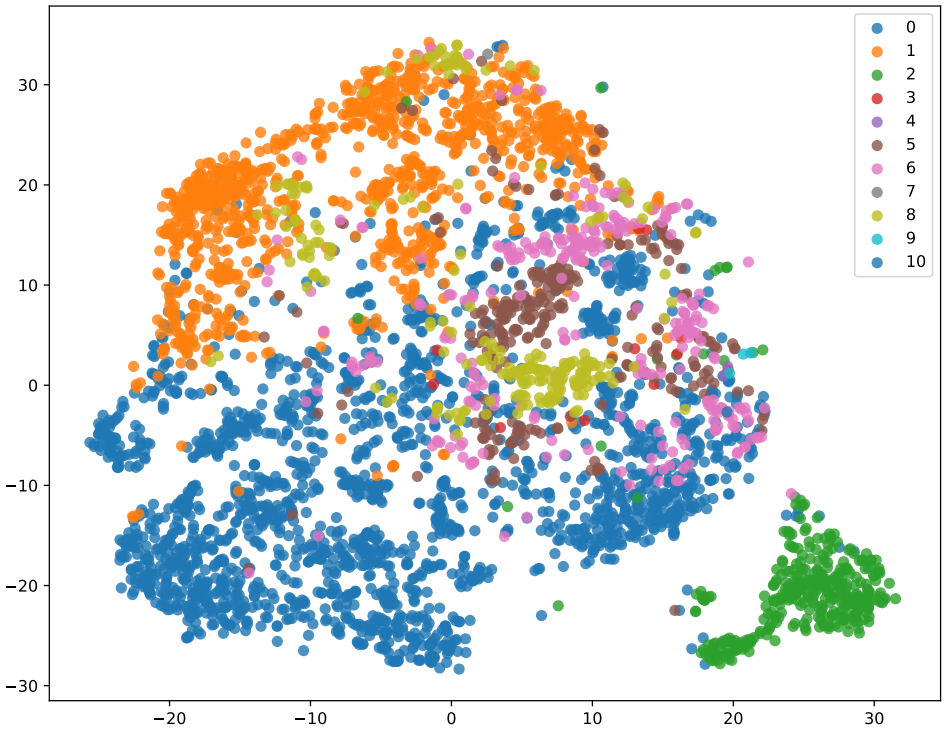


Figure D.2: TSNE projection

Appendix E

Event embeddings (PCA)

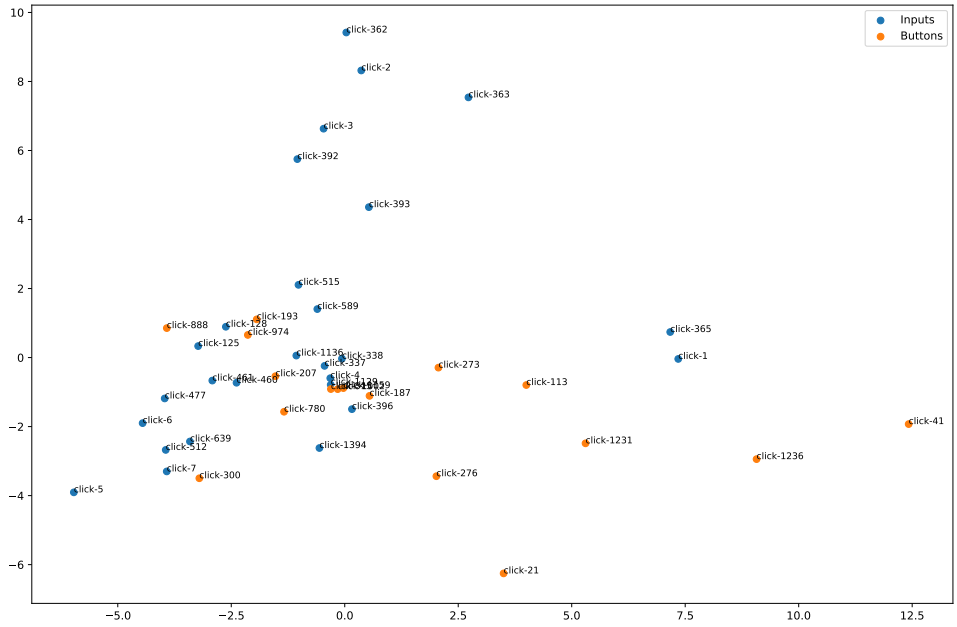


Figure E.1: PCA visualization of inputs and buttons in the application. Those embedings capture quite well semantic structure of the application.

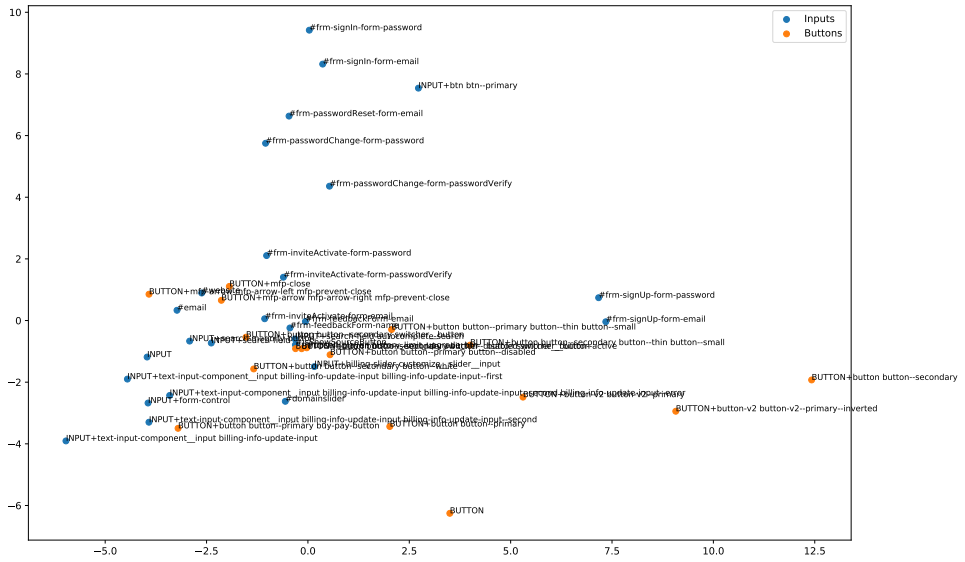


Figure E.2: PCA visualization of inputs and buttons in the application with reduced CSS selectors

Appendix F

Bibliography

[ngr, 2018] (2018). n-gram.

[Chapelle et al., 2010] Chapelle, O., Schlkopf, B., and Zien, A. (2010). *Semi-Supervised Learning*. The MIT Press, 1st edition.

[Chen, 2018] Chen, H.-H. (2018). Behavior2vec: Generating distributed representations of users’ behaviors on products for recommender systems. *ACM Trans. Knowl. Discov. Data*, 12(4):43:1–43:20.

[Deshpande and Karypis, 2004] Deshpande, M. and Karypis, G. (2004). Selective markov models for predicting web page accesses. *ACM Trans. Internet Technol.*, 4(2):163–184.

[Harris, 1954] Harris, Z. S. (1954). Distributional structure. *WORD*, 10(2-3):146–162.

[Hartigan and Wong, 1979] Hartigan, J. A. and Wong, M. A. (1979). A k-means clustering algorithm. *JSTOR: Applied Statistics*, 28(1):100–108.

[Jones et al., 01] Jones, E., Oliphant, T., Peterson, P., et al. (2001–). SciPy: Open source scientific tools for Python. [Online; accessed <today>].

[Jones, 1973] Jones, K. S. (1973). Index term weighting. *Information Storage and Retrieval*, 9(11):619 – 633.

[Jonáš, 2017] Jonáš, A. (2017). Modeling on-line user behavior using url embedding.

[Le and Mikolov, 2014] Le, Q. V. and Mikolov, T. (2014). Distributed representations of sentences and documents. *CoRR*, abs/1405.4053.

[Lu et al., 2006] Lu, L., Dunham, M., and Meng, Y. (2006). Mining significant usage patterns from clickstream data. In Nasraoui, O., Zaïane, O., Spiliopoulou, M., Mobasher, B., Masand, B., and Yu, P. S., editors, *Advances in Web Mining and Web Usage Analysis*, pages 1–17, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Ma et al., 2016] Ma, Q., Muthukrishnan, S., and Simpson, W. (2016). App2vec: Vector modeling of mobile apps and applications. *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 599–606.

[Manning et al., 2008] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[Mikolov et al., 2013a] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013a). Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA. Curran Associates Inc.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

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

[Raghavan et al., 2007] Raghavan, U. N., Albert, R., and Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3).

[Řehůřek and Sojka, 2010] Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. `http://is.muni.cz/publication/884893/en`.

[Rousseeuw, 1987] Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53 – 65.

[Sagar Arora,] Sagar Arora, D. W. Decoding fashion contexts using word embeddings.

[Scott et al.,] Scott, D., T., D. S., W., F. G., K., L. T., and Richard, H. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

[Silahtaroglu and Donertasli, 2015] Silahtaroglu, G. and Donertasli, H. (2015). Analysis and prediction of e-customers' behavior by mining clickstream data. In *Proceedings of the 2015 IEEE International Conference*

on Big Data (Big Data), BIG DATA '15, pages 1466–1472, Washington, DC, USA. IEEE Computer Society.

[van der Maaten and Hinton, 2008] van der Maaten, L. and Hinton, G. (Nov 2008). Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9: 2579–2605.

[Vasile et al., 2016] Vasile, F., Smirnova, E., and Conneau, A. (2016). Meta-prod2vec: Product embeddings using side-information for recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 225–232, New York, NY, USA. ACM.

[Wang et al., 2017] Wang, G., Zhang, X., Tang, S., Wilson, C., Zheng, H., and Zhao, B. Y. (2017). Clickstream user behavior models. *ACM Trans. Web*, 11(4):21:1–21:37.

[Wang et al., 2016] Wang, G., Zhang, X., Tang, S., Zheng, H., and Zhao, B. Y. (2016). Unsupervised clickstream clustering for user behavior analysis. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 225–236, New York, NY, USA. ACM.

[Wei et al., 2012] Wei, J., Shen, Z., Sundaresan, N., and Ma, K. L. (2012). Visual cluster exploration of web clickstream data. In *2012 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 3–12.

[Weller, 2018] Weller, T. (2018). Compromised account detection based on clickstream data. In *Companion Proceedings of the The Web Conference 2018*, WWW '18, pages 819–823, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

[Zhu and Ghahramani, 2002] Zhu, X. and Ghahramani, Z. (2002). Learning from labeled and unlabeled data with label propagation. Technical report.

[Řehůřek, 2013] Řehůřek, M. (2013). Word2vec in python, part two: Optimizing.