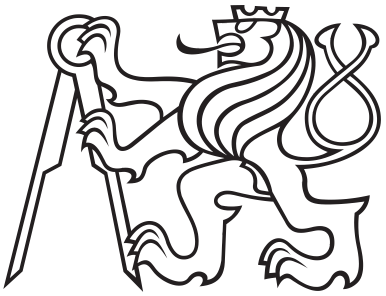
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Control Engineering



Bachelor Thesis

Detection of Anomalies in User Behaviour

Tomáš Vyskočil

Supervisor: Ing. Jan Šedivý CSc.

Study Programme: Cybernetics and Robotics

Specialisation: Systems and Control

May 21, 2015

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Control Engineering

BACHELOR PROJECT ASSIGNMENT

Student: **Tomáš Vyskočil**

Study programme: Cybernetics and Robotics
Specialisation: Systems and Control

Title of Bachelor Project: **Detection of Anomalies in User Behaviour**

Guidelines:

1. Design and implement a statistical model for detecting atypical users of a search engine.
2. Use the provided query logs. Measure the quality of the model using metrics (accuracy, precision, recall) on a provided dataset.
3. Review current state of the art.
4. Analyse the provided dataset
5. Design and implement an anomaly user detection model.
6. Analyse the discriminativeness of the features describing users (behaviour).
7. Measure the detection accuracy of the solution.

Bibliography/Sources:

[1] Fang Yu, Yinglian Xie, and Qifa Ke. 2010. SBotMiner: large scale search bot detection. In Proceedings of the third ACM international conference on Web search and data mining (WSDM '10). ACM, New York, NY, USA, 421-430.
[2] Athena Stassopoulou and Marios D. Dikaiakos. 2006. Crawler Detection: A Bayesian Approach. In Proceedings of the International Conference on Internet Surveillance and Protection (ICISP '06). IEEE Computer Society, Washington, DC, USA, 16-.
[3] Pang-Ning Tan and Vipin Kumar. 2002. Discovery of Web Robot Sessions Based on their Navigational Patterns. Data Min. Knowl. Discov. 6, 1 (January 2002), 9-35.
[4] Junsup Lee, Sungdeok Cha, Dongkun Lee, and Hyungkyu Lee. 2009. Classification of web robots: An empirical study based on over one billion requests. Comput. Secur. 28, 8 (November 2009), 795-802.

Bachelor Project Supervisor: Ing. Jan Šedivý, CSc.

Valid until the summer semester 2015/2016

L.S.

prof. Ing. Michael Šebek, DrSc.
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 15, 2015

Acknowledgments

I would like to thank my supervisor Ing. Jan Šedivý CSc. and his colleague Ing. Tomáš Tunys for the advices they have given me, for their patience, and their sense of humor.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 22.5.2015 Tomáš Vyskočil

Anotace

Internetové vyhledáváče se staly nepostradatelným nástrojem našeho každodenního života. Možnost pohodlně a bez prodlevy vyhledávát informace denně přiláká miliardy lidí. Bohužel někteří uživatelé, lidští či naprogramovaní, se snaží vyhledávače zneužívat ve svůj prospěch, a to například tím, že klikají v nadměrném množství na výsledky vedoucí na jejich doménu, aby zvýšili její popularitu, a tím zlepšili pořadí domény mezi výsledky. Takové chování však může často vést ke zhoršení „uživatelského požitku“ ostatních návštěvníků, a obecně funkcionality vyhledávače. Možností, jak se proti podvodnému a jinému atypickému chování bránit, mají vyhledávače málo, jelikož musí zůstat snadno dostupné. Z důvodu obrovského objemu uživatelů také nepřipadá v úvahu detekovat podvádějící uživatele manuálně, což dále znemožňuje případné natrénování jednoduchého klasifikátoru. Tato bakalářská práce se zabývá způsoby hledání klasifikátoru pomocí metody učení bez učitele, který umožní detekovat toto „anomální“ uživatelské chování. Dohromady ukazuje tři modely využívající různé charakteristiky uživatelských relací. Předběžné výsledky ukazují, že dosažené poznatky by se po dalším rozpracování mohly využít i v praxi.

Abstract

Search engines have become a fundamental tool of everyday live, billions of users are using them to get the information they desire in a comfortable way. Unfortunately, some visitors exhibit various kinds of malicious behavior. For instance, they try to deplete competitions advertising budget through excessive clicking on sponsored results. Such anomalous behavior often leads to a worsened user experience of "normal users". In addition, due to the vast amounts of visitors, such behavior is hard to detect manually, which further means that we can't use standard supervised methods to train a *user* classifier. This thesis introduces three unsupervised models for atypical user detection, all of them evaluate distinct user session characteristics, for instance, click, query and behavioral patterns. The preliminary results show, that with some further improvements the current findings could be deployed for real world use.

Contents

1 Introduction **5**
- 1.1 Thesis goals . . . 5

2 Examples Of Anomalous Behavior **7**
- 2.1 Search engine results scraping . . . 7
- 2.2 Click Fraud . . . 7
- 2.3 Click Spam . . . 7
- 2.4 Other . . . 7

3 Related Work **8**
- 3.1 Distinguishing Humans from Bots in Web Search Logs . . . 8
  - 3.1.1 Approach . . . 8
  - 3.1.2 Conclusion . . . 9
- 3.2 Characterizing Typical and Atypical User Sessions in Clickstreams 9
  - 3.2.1 Markov Chain Model . . . 9
  - 3.2.2 Multidimensional Session Model . . . 10
  - 3.2.3 Characterizing outliers . . . 11
  - 3.2.4 Evaluation . . . 11
- 3.3 Search Engine Click Spam Detection Based on Bipartite Graph Propagation . . . 11
  - 3.3.1 Modeling User Sessions . . . 11
  - 3.3.2 User-Session Bipartite Graph Propagation Algorithm . . . 13
  - 3.3.3 Evaluation . . . 13
- 3.4 SBotMiner: Large Scale Search Bot Detection . . . 14
  - 3.4.1 History Based Suspicous Bot Detection . . . 14
  - 3.4.2 Matrix-based Search Bot Detection . . . 14
  - 3.4.3 Evaluation . . . 15
- 3.5 A Large-scale Study of Automated Web Search Traffic . . . 16

4 Hadoop **18**
- 4.1 HDFS . . . 18
- 4.2 MapReduce . . . 18
- 4.3 Programming Languages . . . 18
- 4.4 Hardware . . . 18

5 Data **20**
- 5.1 Format . . . 20
- 5.2 Labeled dataset . . . 21

6 Proposed Solution **22**
- 6.1 Simple Markov Model . . . 22
  - 6.1.1 Markov Chain States . . . 23
  - 6.1.2 Number of Result Click States . . . 23
  - 6.1.3 Creating the Transition Matrix . . . 25

6.1.4 Session likelihood evaluation . . . 27
6.2 Time Markov Chain Model . . . 28
6.2.1 Markov Chain states . . . 29
6.2.2 Transition Matrix . . . 30
6.2.3 Evaluating the session likelihoods . . . 30
6.3 Multiple Feature Model . . . 33
6.3.1 List of features . . . 33
6.3.2 Evaluating user sessions . . . 35

7 Implementation **36**
7.1 Simple and Time Markov Model . . . 37
7.1.1 MarkovianStateWritable . . . 37
7.1.2 LongPairDoubleWritable . . . 37
7.1.3 MarkovianModelMapper . . . 37
7.1.4 KeyValue pair partitioning and grouping . . . 38
7.1.5 The reduce tasks . . . 38
7.1.6 Python scripts . . . 38
7.1.7 Summary . . . 38
7.2 Multiple Features Model . . . 40
7.2.1 Query and result click data extraction . . . 40
7.2.2 Extraction of the session vectors . . . 40
7.2.3 Covariance Matrix Estimation . . . 41
7.2.4 Evaluating the Mahalanobis Distance . . . 41

8 Evaluation **42**
8.1 Evaluation Stages . . . 42
8.2 A fourth model . . . 42
8.3 Preprocessing . . . 43
8.4 Visual Evaluation and Analysis . . . 43
8.4.1 Results . . . 46
8.5 Evaluation using Metrics . . . 53
8.5.1 Results . . . 53
8.6 Automated Evaluation . . . 55
8.6.1 Testing scenarios . . . 55
8.6.2 Results . . . 56
8.7 Summary . . . 56

9 Conclusion **58**

10 Appendix **61**

List of Figures

1 Visualization of a Markov Chain with transition probabilities . . 24
2 Click rank distributions 24
3 State transition visualizations: row-oriented transitions 26
4 Average Markovian LogLikelihood scores distribution 29
5 Mavg scores 32
6 Markov models flow charts 39
7 Multiple Feature Model shortened flow chart 41
8 Scatter plots 45
9 SMM evaluation 47
10 TMM evaluation 48
11 MFM evaluation 50
12 MM evaluation 52
13 Submit to other state time interval distributions 62
14 Time Markov Model Transition Matrix 63

List of Tables

1 Models Metric Evaluation 54
2 Models Metric Evaluation, second bot feature ignored 54
3 Automated Evaluation Results 56

1 Introduction

Today's Search engines have become a fundamental tool of everyday live, billions of users are using them to get the information they desire in a comfortable way. This vast amount of daily visitors has also helped search engines to become an extremely important advertising channel. Unfortunately those are also reasons why search engines attract users with malicious intentions, who, for example, try to influence the search engine's results ranks, deplete competitions advertising budgets by clicking on sponsored results or use the search engine to find vulnerable web sites. Among other negative consequences, such behavior leads to worse user experience of "normal users".

Users with malicious behavior can be humans or automated programs, and the search engine's defense against them (or consequences of their behavior) is rather complicated, as it has to stay available for all visitors.

This thesis focuses on the topic of *Detection of Anomalies in User Behavior*, multiple examples of such anomalies are listed in Section 2 for illustration. In Section 6 we propose three unsupervised models designed for the task of anomalous behavior detection, which extend previous methods described in related work Section 3. Two of the models are taking the approach of modeling user sessions with Markov Chains. The third model represents user sessions as a multidimensional vector containing 26 features, and then computes the distance from the estimate of the user session vector distributions. Most of the models subtasks were implemented in the Hadoop distributed computing framework, which is briefly introduced in Section 4, more information on the implementation itself can be found in Section 7. After processing more than 3 months of query logs, whose form is described in Section 5, we identified up to 18 million user sessions as anomalous. In evaluation Section 8, we use three distinct methods to verify the models performance, which were chosen based on our setting of unsupervised learning. Nevertheless, after evaluating all the models, we agreed upon that the Markov Chain based models are not suitable for real world use. The third model has shown the best and most promising performance. However, as of now, this model also can't be included into the search engine's workflow and has to be further improved. Among the possible future improvements are, for instance, a better density estimation such as decision trees, or use of location based features. All things considered, this thesis provides a solid springboard for future work.

1.1 Thesis goals

- Design and implement a statistical model for detecting atypical users of a search engine
  - The proposed solution can be found in Section 6, the implementation in 7
- Use the provided query logs. Measure the quality of the model using metrics (accuracy, precision, recall) on provided dataset.

  - The quality of the models is evaluated in Section 8

- Review current state of the art
  - The current state of the art is presented in Section 3
- Analyze the provided dataset
  - Multiple views on the dataset are presented throughout the thesis, particularly in Sections 6 and 8
- Design and implement an anomaly user detection model
  - The design is being discussed in Section 6, the implementation in 7
- Analyze the discriminativeness of the features describing users behavior.
  - The use of the proposed features is substantiated in Section 6, Section 8 shows the consequences of use of the features
- Measure the detection accuracy of the solution
  - The accuracy and other metrics are evaluated in Section 8

2 Examples Of Anomalous Behavior

Before we proceed to the main content of the thesis, we would like to show a few examples of possible anomalous user behavior. In most cases such behavior is expected to be executed by automated programs (bots), however all of the examples listed in this section could also be conducted by human users.

2.1 Search engine results scraping

As research shows [3] the rank of a result on the search engine results page (SERP) has a substantial importance. Some website owners and SEO (Search Engine Optimization) companies thus use various "scraping" tools, which automatically check their or competitions website ranks. Nevertheless this is only one of multiple possible reasons for results scraping.

2.2 Click Fraud

Another example of possible atypical (and malicious) behavior is click fraud, which is a practice of deceptively clicking on search ads with the intention of either increasing third-party website revenues or exhausting an advertiser's budget [17].

2.3 Click Spam

Click Spam is a method used by website owners or SEO companies which aims to improve webpage's rank in search engine results. Search results are ranked using multiple algorithms (e.g. the PageRank [11]) and criterions, user result clicks are [7] among them. In addition, researches focusing on search engine users behavior [3] show that users are more likely to click on a result if its rank is higher. As a consequence, cheating website owners are trying to increase the number of clicks illegally with the use of automated programs.

In [10] the authors present the following click spam examples:

- Repeated URL clickers - after submitting a query,the "user" clicks the same result repeatedly
- Same domain clickers - after submitting different queries, the "user" clicks on results with the same domain repeatedly

2.4 Other

In [18] the authors list multiple types of malicious user behavior. Besides Click Fraud or Click Spam, they show that some bots are using search engines to find website with security vulnerabilities, or to harvest emails for spam. Another well known anomalous behavior is *Google Bombing*, which might target to influence the search engine query autocomplete system.

3 Related Work

Considering that the solution proposed in this thesis is strongly based on previous work, the Related Work section pays more attention to previous methods than usual, even though a brief introduction might typically suffice.

During recent years, many scientific papers were published on the topic of atypical user detection. However, some approaches are not feasible in our case as our dataset misses some of the needed features. For example we do not have access to results of CAPTCHA tests, which is used in [9] to create a base labeled dataset that is later used to train a bot detection model.

To sum up, the works introduced in this section can be separated into the following three categories:

1. Simple statistics approach: represented by [5]
2. User sessions model approach: represented by [10] and [13]
3. Distributed bot detection approach: represented by [18]

In addition to these three categories, we present an analysis of various user characteristics from [2].

3.1 Distinguishing Humans from Bots in Web Search Logs

This work [5] focuses on cleaning web search logs from bot activity. The researchers aim to assess the impact of bot traffic on the search engine's services and through filtering out non-human activity they aim to improve characterization of human search behavior. Analyzing a log from AOL corporation which includes 3 months of search data from 2006, they identified 92% of users as human and 0.6% as bots.

3.1.1 Approach

The authors did not have access to labeled data and therefore they introduce a system of multiple criterions (e.g. number of repeated queries.) that distinguishes between humans and bots. In order to make the approach of criterions as robust as possible they use two thresholds for each of the criterions. The first threshold is being used to recognize users who are most probably human, the second is used to recognize users who are most probably bots. Both thresholds are set so that they avoid ambivalence. In addition, one more threshold (called strong criterion) which represents values that are not achievable for human users (e.g. user repeated the same query 20000 times) is introduced. Note that this approach splits users in these three classes: human, bot or unknown.

Among the used criterions are:

1. Queries per day
2. Queries per minute

3. Minimum time interval between queries
4. Maximum number of repetitions of the same query
5. Number of repetitions with precise (time) repetition
6. Maximal continous session length

An user is classified as human/bot only if he was classified as human/bot by all criterions. If this rule did not yield a final result, but the user exceeded one of the strong criterions, he is labeled as a bot.

3.1.2 Conclusion

The researchers have agreed that building a method for human-bot classification without labeled data is very difficult and that their approach has multiple disadvantages. Notably setting up the thresholds so that they separate the human and bot classes is often problematic or even impossible. Also the method leaves 3-7% of users unclassified, because the bots frequently exhibit extreme behavior according to some criterions, but act as human user according to another one.

3.2 Characterizing Typical and Atypical User Sessions in Clickstreams

Characterizing Typical and Atypical User Sessions in Clickstreams [13] (CTAUSC) is a paper which is targeting for improvement in estimation of search engine user experience metrics such as Click Through Rate (ratio of query result views to result clicks). To improve robustness of data mining techniques applied to web search logs, the researchers claim that one has to separate typical and atypical user sessions in clickstreams. To achieve this, the authors propose a method which models user sessions as a sequence of actions and then identifies outliers in *user session space* using Mahalanobis distance. The method namely comprises of the following parts:

1. Building a Markov Chain Model
2. Building a Multidimensional Session Model
3. Characterizing outliers

3.2.1 Markov Chain Model

Every user session is split in *Event-Locality Pairs* (ELP) consisting of an ordered pair in form of (event, page number). These event pairs represents some sort of user's action. The aforementioned events are defined as follows:

- Page request (marked as P) - query submit to the search engine
- Web click (W) - click on results

- Sponsored Click (O) - click on sponsored links
- Next Click (N) - click on next page button on SERP
- Any Click (A)- click that does not belong to W,O or N

After all user sessions from the dataset have been assigned their ELP representation, a Markov Chain model can be built. The Markov Chain model states are directly represented by the ELP pairs and the state (ELP) transition probability from state i to state j is estimated as follows:

$$Pr(i,j) = \frac{Q_{i,j}}{Q_i} \tag{1}$$

where $Q_{i,j}$ is the number of transitions from state i to state j. And $Q_i = \sum_j Q_{i,j}$. Transitions with high probability are considered as normal behavior, whereas transitions with low probability as atypical.

Once the probability of all ELP transitions has been estimated, all user sessions can be assigned a likelihood score. In order to prevent underflow and bias to assign lower scores to longer user sessions, a Markovian Average LogLikelihood (Mavg) is being used.

$$M_{avg} = \frac{\ln \mathcal{L}}{S} \tag{2}$$

where $\mathcal{L}$ is the session likelihood and S is the number of ELP transitions in session.

3.2.2 Multidimensional Session Model

The above mentioned model is indeed capable of identifying some of the atypical user sessions, but its disadvantage is that a very similar likelihood score can be assigned to considerably different user sessions. This means that some other user session characteristics need to be incorporated into the model. The authors have therefore decided to include the following features in to the final model:

- Mavg: The maximum likelihood avg score from the Markov Model
- P_t: Number of Page Requests
- W_t: Number of Web clicks
- O_t: Number of Sponsored clicks
- N_t: Number of next clicks
- A_t: Number of Any clicks
- $E = P_t + W_t + O_t + N_t + A_t$: Total number of events.

The resulting 7 dimension vector characterizing a user session is represented as:

$$q = (M_{avg}, \frac{P_t}{E}, \frac{W_t}{E}, \frac{O_t}{E}, \frac{N_t}{E}, \frac{A_t}{E}, E) \tag{3}$$

3.2.3 Characterizing outliers

The Mahalanobis distance (d) is computed for all user sessions as follows:

$$d = \sqrt{(q-\mu)\Sigma^{-1}(q-\mu)^T} \tag{4}$$

where μ is the mean session vector and Σ is the covariance matrix for the clickstream characteristics in the 7 dimensional space and q is the user session vector with log-transformed dimensions. The distribution of the user session model is estimated as the multivariate Gaussian distribution. The trailing 1% of user sessions is then marked as atypical and further studied.

3.2.4 Evaluation

The evaluation consisted of manual testing on a sample of 100 user sessions. First half of these sessions was randomly sampled from the atypical sessions, the other half from the "normal" sessions. This sample was given to a panel of three unbiased human judges.

As a result, the judges have agreed on 72% of sessions. 89% of the sample atypical sessions were labeled by the majority of the judges as atypical, whilst 88% of normal sessions were labeled as normal. Filtering out the atypical sessions reduced the CTR mean uncertainty by 40%.

3.3 Search Engine Click Spam Detection Based on Bipartite Graph Propagation

This paper focuses on detection of *Click Spam*, which is a method aiming to promote a websites rank in the search engine results.

The authors take a very similar approach to that in [13], thus they model user sessions with sequence of actions which allows them to detect spam by looking for abnormal sequences. They state that the previous models did not take into consideration the semantic meaning of the user actions e.g. two consecutive submits of different queries vs consecutive submits of different queries. Their model takes this semantic meaning into consideration, and also adds time as a factor in the model.

3.3.1 Modeling User Sessions

Similarly to the previous work CTAUSC, the authors propose the following user actions:

- Q_i - query submit to the search engine, in which i is used to denote different queries
- W_i - click on results, i denotes different web clicks
- O_i - click on sponsored links, i denotes different sponsored links

- N - Load a new page, including clicking on the next page, the previous page and specific page numbers
- T - scroll a page
- A_i - clicks on other elements than those previously proposed, including the Images or Videos tabs

After defining the user actions, they introduce time intervals (denoted as T) in to the model as follows:

- T_0: zero time interval "t"
- T_1: $0 < t \leq 10$ seconds
- T_2: $10 < t \leq 30$ seconds
- T_3: $t > 30$ seconds

These actions represent the Markov Model States and the transition probability from state i to state j is defined followingly:

$$Pr(i, j) = \frac{Q_{i,j}}{Q_i} \tag{5}$$

where $Q_{i,j}$ is the number of transitions from state i to state j. And $Q_i = \sum_j Q_{i,j}$. After defining the sessions states and calculating all the state transitions probabilities, the user sessions can be assigned their Maximum Likelihood Average scores. User sessions with Mavg less than -4 are labeled as cheating sessions. As a result, 99.6% of sessions were labeled as normal, and 0.4% as atypical.

The researchers then conducted a further analysis of the cheating sessions, which revealed the following cheating session modes:

- (QA_i) - user submits different queries, but clicks always the same domain repeatedly
- (QT_i) - user submits different queries and then scrolls the page repeatedly
- Q_i- submit same query repeatedly
- $Q(W_i)$ - submit a query, click the same result after a very short period of time
- $Q(A_i)$ - submit a query, click the same domain after a very short period of time

If a subsequence of a user session that matches one of the cheating modes exceeds 50% of length of the session, the session is also labeled as cheating session.

3.3.2 User-Session Bipartite Graph Propagation Algorithm

This algorithm is build upon assumption that if a user conducted some cheating sessions, then all his other sessions are likely to be cheating sessions.

First, each user session is assigned a score, cheating sessions get score 1 and all others 0. Then each user is assigned a score, which equals to the weighted average of all his sessions. Then every session is assigned a new score, which equals to weighted average of all users who have made such session. This procedure of evaluating score of sessions and users is repeated until the change in score between consecutive iterations is insignificant.

After the procedure has finished, the authors have obtained a list of cheating sessions and cheating session sequences. The cheating session sequences list is further used as an input to the *Pattern-session bipartite graph algorithm*, which searches for cheating sequences in the remaining "normal" sessions and thus produces the final list of cheating sessions.

3.3.3 Evaluation

The authors have used a query log from a *popular Chinese commercial web search engine* from December 2011 for evaluation.

The researchers have used manual annotation on a sample of 100 sessions to evaluate the precision of cheating session detection based on the cheating mode presence. As a final result, this detection has a precision of 98.6%, and reveals about 40000 cheating sessions.

The User-session Bipartite graph algorithm labeled 2.1% of sessions as click spam, whilst the Pattern-session bipartite graph algorithm 2.6%. The precision of both was evaluated using manual annotation on a sample of 100 cheating sessions.

To further evaluate the performance of the proposed algorithm, the researchers calculated the NDCG metric (of *ranking quality* or in other words search result relevance) before and after filtering out the click spam sessions. The NDCG (Normalized discounted cumulative gain) at particular position p is defined as follows:

$$DCG(p) = \sum_{k=1}^{p} \frac{2^{rel_i} - 1}{log_2(1+i)} NDCG(p) = \frac{DCG(p)}{IDCG(p)} \tag{6}$$

Where rel_i is the manual label of the i-th document, IDCG is the ideal value of DCG when sorting results according to relevance.

Note that the query results were sorted according to their click-through rate from the original query log and from the filtered one before computing and comparing NDCG.

For query results whose rank was affected by click spam, this improved the NDCG by up to 5%.

3.4 SBotMiner: Large Scale Search Bot Detection

In contrast with the above presented approaches, which focused on individual atypical user detection, SBotMiner is a system for automatical identification of stealthy, low-rate search bot traffic from query logs [18].

The processing flow of the SBotMiner's approach consists two steps, the suspicous bot detection and matrix-based bot detection.

3.4.1 History Based Suspicous Bot Detection

Bots in general can have various kinds of motivation, however motivations like 'promoting webpage's pagerank' or 'depleting competitor's ' advertising budgets can only be achieved through excessive query submitting and/or SERP result clicking. This implies that in order to detect suspicious bot activity, one has to look at queries or results whose popularity has risen rapidly recently.

Result-click histograms are built to detect queries whose current popularity has grown recently. A *result-click* histogram consists of bins that represent a result R which is presented to the user after he submits query Q to the search engine. The "height" of a bin is given by number of clicks on the result. For every query Q two such histograms are created. One from older query log data (e.g. previous month) and another one from current logs (current month). These histograms are further compared, and if the histograms differ significantly, the given query is labeled as suspicious. Also queries which were submitted significantly more times than in previous period are added into the suspicious group.

Note that these queries don't include only suspicious activity, but they also include current "trends", e.g. gifts during pre-Christmas time or death of a celebrity. This means that the users who have submitted these queries can be both bots or "flash crowds". In order to distinguish between these groups a matrix based approach is used which is described in the next section.

3.4.2 Matrix-based Search Bot Detection

This bot detection approach is build upon an assumption that bot behavior is controlled by a script and therefore must be less diverse than normal user behavior.

For every suspicious query, the users who have issued this query are selected, and all queries submitted by these these users are extracted in order to build a "Mqc" matrix. Each row of this matrix represents a query and each row represents a user. For instance M(i,j) represents the number of times the query i was submitted by user j.

For every suspicious query, all users who have issued this query are selected to form a group G. Then all queries submitted by users in group G are extracted in order to build a "Mqc" matrix. Each row of this matrix represents a query, each row represents a user. For instance M(i,j) represents the number of times the query i was submitted by user j. Once this matrix has been built a "focusness" of the group is computed as follows:

$$F_{qc} = \frac{\Sigma_j M_{qc}(q,j) | \forall j, s.t. \Sigma_{i \neq q} M_{qc}(i,j) = 0}{\Sigma_{ij} M_{qc}(q,j)} \tag{7}$$

In other words, the focusness of a group G simply represents the ratio of traffic originating from G to the whole traffic in G. After evaluating the focusness of all groups with above described equation the researches have found out that over 70% of the groups have Fqc¿ 0.7, and at least 70% of users in these groups do not submit other queries than the suspicious one.

Those groups whose F_{qc} is greater than 0.9 are labeled as bot groups. This threshold was selected rather conservatively in order to prevent false positives, which implies that it may overlook some of the malicious users. In addition the simple threshold approach ignores bots who simulate some normal behavior. To detect such users, the PCA algorithm is used.

PCA or Principal Component analysis is one multiple methods of projection high dimensional data onto lower dimensional space, what separates it from other methods is that it seeks a projection which fits the data best in least square sense [4].

Bots might simulate some normal (and random) behavior but according to the researchers, they often use a similar set of queries. The PCA algorithm is therefore used to identify correlated user query patterns. After obtaining the largest principal component P1, the original data is projected on the subspace defined by P1. This means that by projecting the vector u of some user, we obtain a vector u'. For every vector pair u,u' the L2 norm difference $||u - u'||$ is evaluated. Note that the L2 norm is defined as follows [15]:

$$|x| = \sqrt[2]{\sum_{k=1}^{i} x_k} \tag{8}$$

The vectors whose L2 norm difference is "very small", which means that the its projection into the one dimensional sub space did not change the vector significantly, are labeled as suspicious user vectors.

3.4.3 Evaluation

The researchers have used two months of query logs from February and April 2009. The SBotMiner method identified approximately one million suspicious groups and the Matrix based detection approach has labeled 600 thousand of them as bot groups, which accounted for roughly 3.8% of the user traffic. The PCA detection approach alone identified 137 groups as bot groups, 60% of these groups have had more than 2000 users.

The validation consisted of checking the presence of additional features of the users. These features included hash of cookie, cookie creation date, IP address, hash of user agent. If 99% of pageviews of bot labeled group agreed on one of the features, the group is considered to have one identical feature. More than 95% of the groups have agreed on 2 and more features. Only 0.3% of the groups did not agree on any feature, thus they are considered as false positives.

3.5 A Large-scale Study of Automated Web Search Traffic

In [2] the researchers propose two classes of potential features for detecting automated search engine traffic.

The first class represents human physical limitations e.g. the typing speed and includes the following features.

- Number of Requests, Queries Clicks: The authors give an example of a distribution of number of query submits per day. They state that a human user with 200 queries in a day is possible but very unlikely, and most users with this volume of submits (or more) appear to be automated.
- Query rate: The researchers show an example that humans rarely submit more than 7 queries in 10 second interval.
- Number of IP addresses / Locations The analysis shows that some users change their geographical position in a very rapid manner. Whilst it is possible that e.g. mobile users change their position or IP address often, huge variances in position are unlikely.

The second class of potential features takes in account the fact that automated search traffic might mimic human behavior. To improve accuracy of classifying human/automated traffic the following features are proposed:

- Click-through Rate (CTR or submits/clicks ratio): The authors state that previously published human CTR vary, but most show that human user clicks at least once in ten queries. They also introduce three common types of automated users: those who click at all presented links (most common type), a bot that does not click and bots that click only on a targeted link.
- Alphabetical Ordering of Queries According to the researchers, some bot issue queries in significant alphabetical ordering, which can be easily detected.
- Spam Score To evaluate this feature, a set of ¡spam word, weight¿ is being used to calculate the spam score of queries of a user.
- Adult Content Score Although adult content is indeed a usual target for human users, a excessively high ratio of adult content related queries is considered as atypical behavior and possibly automated.
- Query Keyword Entropy The authors show that many bots submit extremely redundant queries and thus have above normal query keyword entropy. On the other hand some bots issue only one query keyword and so they have zero keyword entropy.
- Query Keyword Length Entropy This feature is proposed because some bots enter only some specific classes of words and have a very low word length entropy in comparison with a human.

- Query Time Periodicity Entropy Bots often send request to the search engine at regular intervals e.g. 5 or 15 minutes. Therefore the authors suggest computing the query time delta entropy of the users e.g. with minute or hour precision.
- Advanced Query Syntax Search engines typically provide multiple query prefixes called operators which allow the user to restrict results according to his needs. For example -filetype operator is used to show results with specified file type e.g. PDF. The researchers show that only 0.1% of users in their sample use more than 5 operators in one day.
- Category Entropy A similar feature to Adult Content and Spam scores. The researchers use category hierarchy to assign a category to each query. Then the category entropy is evaluated for each user according to his queries.

In addition to the above listed features, the authors propose a direct approach to detect bot activity. This includes blacklisted IP addresses, blacklisted user agents or blacklisted country codes.

4 Hadoop

Apache Hadoop is a framework that allows for distributed processing of large data sets across clusters of computers using simple programming models [6]. A more detailed description of the Hadoop framework and architecture is out of the scope of this work, so we focus on the most important features instead.

4.1 HDFS

The first feature is the Hadoop Distributed File System (HDFS), which allows us to store large files (typically gigabytes or terabytes of data) across thousands of computers and therefore provides high throughput (through parallel read) and reliability (through file replication). HDFS also allows to move computations near data, instead of moving data to computations, which significantly reduces unnecessary file traffic.

4.2 MapReduce

Another important feature is computation parallelization, which is achieved by the MapReduce programming paradigm. Map/Reduce is a programming paradigm that expresses a large distributed computation as a sequence of distributed operations on data sets of key/value pairs [6]. MapReduce comprises of two main phases called Map and Reduce. During the Map phase the input data is split into multiple parts which are later assigned to user defined map tasks. Typically for every input file a separate map task is created, whose map function transforms the input file into Key-Value pairs. The Key Value pairs are further grouped by keys, creating tuples in form of Key, Ordered List of Values and these tuples are partitioned into multiple reduce tasks.

The reduce phase invokes the user defined reduce function, which processes the input tuples and creates the final output Key, Value pairs.

4.3 Programming Languages

The default programming language for Hadoop is the Java programming language (late 6 version or higher, as of spring 2015). However the framework allows programmers to use other programming languages through Hadoop Streaming, which is a utility that allows us to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer [6]. This means one could use, for instance, the Python programming language if they prefer it over Java.

4.4 Hardware

According to Apache, Hadoop is build in a way that it can run on commodity hardware. Even though this statement indicates that there is no need to have access to some "custom supercomputers", the definition of "commodity hardware" is unclear. For instance, the company Cloudera [1], which is a Hadoop-based

software provider, recommends the following hardware for "light data processing":

- Two hex-core CPUs
- 24-64GB memory
- 8 disk drives (1TB or 2TB)

For the purpose of this work, we use machines provided by Metacentrum [19], the Czech national grid operator. As of 1st February 2015, the provided hardware specifications were as follows:

- 27×16 cores
- 27×128 GB of RAM
- 1 PB of storage in HDFS

5 Data

The dataset used in this work was provided by a czech SE with more than 6 million daily unique visitors and consists of search logs containing all traffic between 1st September 2014 and 12th December 2014. The data is split in multiple files by days totaling 103 files and over 1.5 billion lines, consuming approximately 1.5 terabytes.

5.1 Format

For every query submit or SERP page request a new line was created, all lines contain multiple fields stored in JSON format in the following form:


```
{"query":"query",
"user":{"id3":"a04df48bf91f446c",
"session":"0f7823d37df0b9ad",
"id2":"7b60bd48ff889b46",
"id":"42da379c71a0b963","created":1398149066},"
autocompleted":false,
"time":1417429632.35,
"ip":"2d215e9e405b268e",
"rss":true,
"js_events":6,
"offset":0,
"ua":"b8a849e7a1478d0b",
"limit":10,
"other_clicks":[1417794297.35]
"result":[{"url":"http://www.somepage.cz/","clicks":[1417429634.58]},
{"url":"http://some-url1.cz/"},
{"url":"http://some-url2.cz/"},
{"url":"http://some-url3.cz/"},
{"url":"http://www.some-url4.cz"},
{"url":"http://www.some-url5.eu/"},
{"url":"http://www.some-url6.eu/"},
{"url":"http://www.some-url7.cz"},
{"url":"http://www.some-url8.cz"},
{"url":"www.some-url9.cz"}]}
```


The meaning of the fields follows:

- user: contains all user related data
  - session: user session unique identifier, is created upon user's visit and is destroyed after the user's browser session ends, is stored in http cookie, is always present
  - id: user's unique identifier, is created upon user's visit, stored in http cookie, is always present

  - id2 and id3: other user's unique identifiers
  - created: timestamp of user id field creation

- autocompleted: true if the query was submitted using the query autocompletion feature, false otherwise
- js_events: contains number of JavaScript events which occurred after the SERP page loaded, a javascript event occurs when e.g. the visitor's mouse hovers over a button or a result
- offset: is used to indicate the SERP page number, the first page has offset equal to 0, the second page 10, third 20 and so on.
- ua: visitor's user agent hash
- rss: if present and true, the user has submitted the query via the rss version of the SE's web
- limit: number of requested results per SERP page, the default limit is 10, if the value is different from 10
- result: query results
- URL: full result URL
  - clicks: contains the click(s) timestamps(s), in case the user clicked on the given result
- other_clicks: timestamps of clicks other than result clicks

5.2 Labeled dataset

As was the case of most previous attempts presented in Related Work section, we do not have access to any labeled data. However we can identify some binary features of users which can help us to infer whether he/she/it is a (most likely) human or (most likely) bot. These features are used for evaluation in Section 8.

6 Proposed Solution

In this section we are going to discuss our Atypical User Detection models. As we have already stated in chapter 2, this model will be based on previous works (all of them are also presented in chapter 2).

In this thesis we are going to implement the following atypical user detection models:

1. Simple Markov Chain Model
2. Time Markov Chain Model
3. Session Feature Model

The first common characteristic of all the above listed models is that they are unsupervised, as we do not have any labeled data.

Second common characteristic of all our models is that they operate on user session level. In our case a user session starts when a user submits a query to the search engine. All consecutive events, including for instance result clicks or next page requests, belong to the session. If a user submitted a new query after more than 30 minutes from the first query, a new session starts.

Third common characteristic of the models is that they ignore "short" sessions. A short session is a session which consists of only one single event. This event is typically a single SERP page request, which is not followed by any clicks or other requests. Such sessions simply don't contain enough information to be evaluated. The decision of ignoring these sessions seems "natural", as their "shortness" could be judged as atypical on its own. Unfortunately short sessions form 30% of all sessions in our dataset and therefore labeling them as "atypical" without further research would not be substantiated. However we can easily find an example of a malicious (human or automated) users who could behave in such way, e.g. a bot which tries to lower competitions website CTR. In brief, dealing with the short session problem is out of scope of this work, and is an important area for possible future study.

Let's now discuss the differences between the models. The first two models are very similar, they both represent user sessions as a sequence of actions and seek to find unlikely session sequences. In contrast with representing sessions as a sequence of events is the approach taken by the Session Feature Model, which tries to represent user sessions as a vector of behavioral features.

6.1 Simple Markov Model

The first of three proposed user session models is the Simple Markov Model. This model is very similar to Markov Chain Model in [13], which is well described in Chapter 2.

In brief this model represents user sessions as a sequence of actions, which directly represent states of the Markov model. Each session in the dataset gets a likelihood score, sessions with low likelihood are considered as atypical.

The hypothesis of this approach is that a normal user session should consist of a logical sequence of actions, whereas atypical user sessions should contain repetitive or even nonsensical patterns.

6.1.1 Markov Chain States

For the user session modeling we use the following Markov Chain states:

- Submit (S): This state represents a submit to the search engine. In comparison from the previous work, we distinguish query submits from page requests.
- Autocompleted Submit: Represents an autocompleted submit to the search engine. This state has never been used in previous implementations. We expect that simple bots do not use the autocompleted feature and this state might help to identify such bots.
- Next Page Request (N): Represents a next page request on the SERP page.
- Result Click 1 to 20 (Ri): Represents a state (Result Click, Result Rank). We distinguish between all clicks on page 1 by states Result Click 1-10, all clicks on page 2 and higher are represented by states Result Click 11-20. For instance click on result two on page 3 or 4 is represented as Result Click 12.
- Other Click (O): Represents other clicks on the SERP page than the result clicks e.g. sponsored result clicks or "images" tab clicks. The previous implementations were able to distinguish between sponsored clicks and any other clicks, unfortunately our dataset misses such feature.
- IP change (I): Represents an IP change during the user session. This state was never used in previous works. There is a potential risk that this state might penalize mobile users, as our dataset does not distinguish between mobile and desktop access. However we expect the rate at which mobile users change their IP addresses to be much lower than the ip change rate of some bot classes.

In total our model distinguishes between 25 states and thus 625 state transitions, including the transition from IP change to IP change, which never occurs.

An example sequence might be [S, R1, O, A], which means that a user submitted a query, clicked the first result, then clicked a sponsored result and then submitted a query using the autocomplete feature.

6.1.2 Number of Result Click States

One of the question raised when creating this model was how many result click ranks to include in the final model. To assess that, we have studied the result

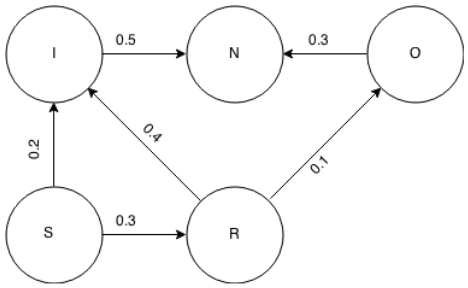


Figure 1: Visualization of a Markov Chain with transition probabilities

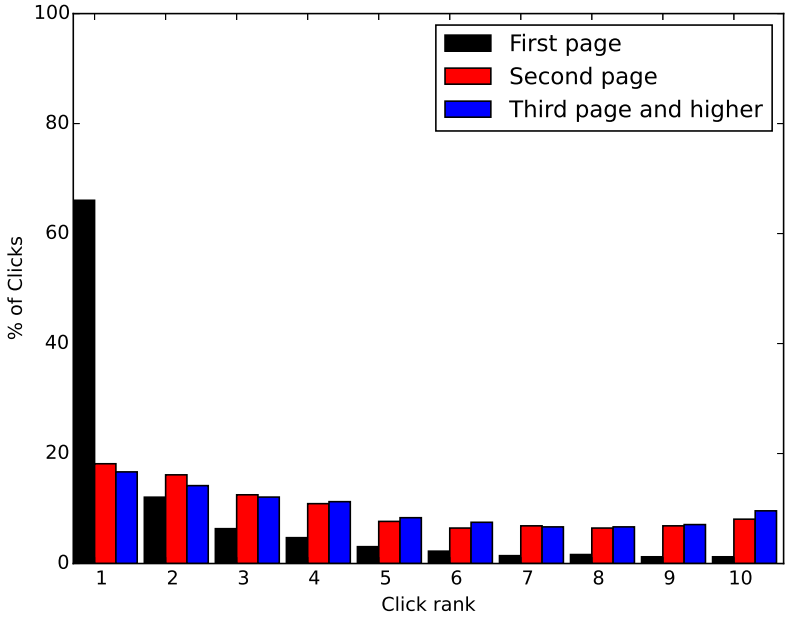


Figure 2: Click rank distributions

click distributions of the SERP pages. In the following Figure you can see the result click distribution on SERP page 1 and 2, and SERP pages 3 and higher.

From the Figure we can easily see that the result click distribution on the first page is substantially different from the click distributions on other pages and thus it is very reasonable to distinguish between clicks on first page and other pages. However when we study click distributions on page 2 and higher pages, we can see that they seem to be much more "uniform". Another key point is that SERP 2 and 3+ distributions look very similar, as a result, we have decided to include all SERP 1 clicks to the model (10 Markov states). Clicks on second

and higher pages are grouped in the model (another 10 Markov states), which yields 20 result click states in total.

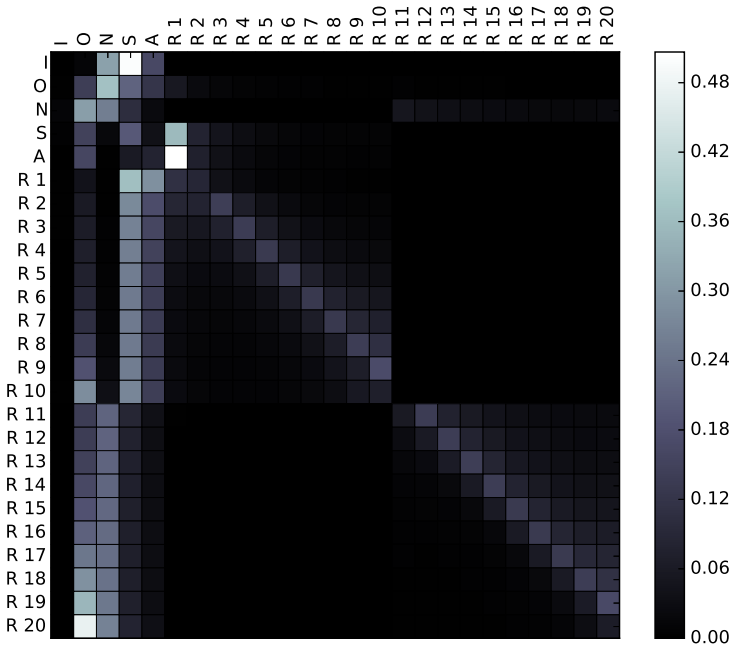
6.1.3 Creating the Transition Matrix

In order to build the Simple Markov model, we need to extract the transition matrix from our data. Firstly, we need to build the state sequence representation of all sessions. Secondly we have to evaluate the number of transitions between all state pairs. After that, the transition probability from state i to state j is estimated as follows:

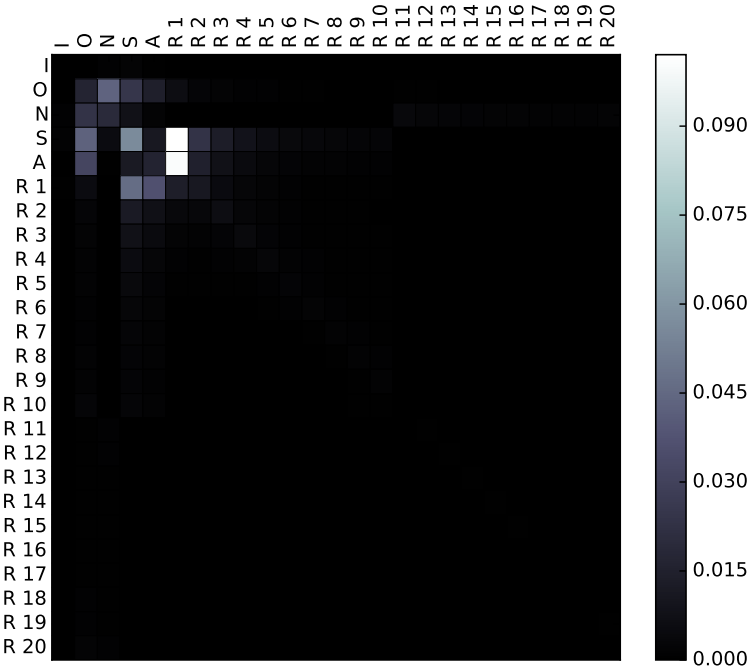
$$Pr(i,j) = \frac{Q_{i,j}}{Q_i} \tag{9}$$

where $Q_{i,j}$ is the number of transitions from state i to state j. And $Q_i = \sum_j Q_{i,j}$. The creation of the transition matrix is implemented in the *Simple Markov Model Builder* Hadoop job, see Implementation section for more information.

The top image in Figure 3 includes a visualization of the resulting transition matrix with rows normalized according to Equation 9. Note that the visualization is row-oriented, for instance the transition probability between an IP change and a submit lies on row 1 column 4.



(a) Markov Model Transition Matrix



(b) All-transitions-normalized Markov Model Transition Matrix

Figure 3: State transition visualizations: row-oriented transitions

We can see that the transition with the highest probability (brightest color) is transition from auto-completed submit(A) to first result click (R1), which is an expected behavior. Another expected feature is that result clicks on first page are more likely to be followed by a new submit than a next page request, whereas a result click on higher pages are more likely to be followed by a next page request.

The bottom image in Figure 3 includes a visualization of the transition matrix with the following normalization:

$$M(i,j) = \frac{Q_{i,j}}{Q} \tag{10}$$

where $Q_{i,j}$ is the number of transitions from state i to state j. And $Q = \sum_i \sum_j Q_{i,j}$. This means that the respective numbers of transitions are normalized by sum of all transitions and thus the visualization enables us to see the most frequent transitions in general.

We can easily see that most state transitions happen in the upper-right 6×6 sub-matrix. Among the most frequent transitions (portrayed with brightest colors) were:

1. autocompleted submit, result click 1
2. submit, result click 1
3. submit, submit
4. result click 1, submit
5. submit, other click
6. result click 1, autocompleted submit
7. autocompleted submit, other click

6.1.4 Session likelihood evaluation

After extracting the Markov Model transition matrix, we can proceed to evaluating the likelihood of all sessions. The probability (likelihood) of observing a Markov chain path (session) $X_{i\,i=1}^{N}$, where N is the number of states, is defined as follows [14]:

$$P(X_1, \cdots, X_N) = P(X_N | X_{N-1}, \cdots, X_1) \cdot P(X_1, \cdots X_{N-1}) \tag{11}$$

Because Markov Chains are memoryless (the next state depends only on the current state, and not on the sequence of states that preceded it), we know that:

$$P(X_N | X_{N-1}, \cdots, X_1) = P(X_N | X_{N-1}) \tag{12}$$

By inserting 12 into 11 we obtain:

$$P(X_1, \cdots, X_N) = P(X_N|X_{N-1}) \cdot P(X_1, \cdots, X_{N-1}) \tag{13}$$

By repeatedly inserting 12 into 13, we obtain:

$$P(X_1, \cdots, X_N) = \prod_{i=1}^{N} P(X_i|X_{N-1}) \tag{14}$$

Where X_0 is the initial state of the markov chain and $P(X_i|X_{N-1})$ are elements of the transition matrix (transition probabilities).

As proposed in [13], we use the Average Markovian LogLikelihood (Mavg) of the sessions in order to prevent underflow and bias to assign lower scores to longer sessions. We also multiply the result by -1 to get positive values. Mavg is then defined as follows:

$$M_{avg} = -\frac{\mathcal{L}}{S} \tag{15}$$

where S is the number of state transitions in session and $\mathcal{L}$ the session likelihood, which is defined as follows:

$$\mathcal{L} = \sum_{i=1}^{N} log(P(X_i|X_{N-1})) \tag{16}$$

Where $P(X_i|X_{N-1})$ are elements of the transition matrix (transition probabilities).

The resulting Mavg score distribution can be seen in Figure 4. Even though our model is slightly different from the one proposed in [13], a comparison between our and their model distributions could be beneficial. Unfortunately the authors did not provide a full plot of Mavg score distribution and thus we can't directly compare it to the Mavg distribution of our model. However they state that 99.6% of sessions have had Mavg <2, in comparison with 66% of sessions in our case. They also state that all sessions have had Mavg<14, which is in conformity with our model.

6.2 Time Markov Chain Model

A natural extension of the Simple model is the Time Markov Chain Model, which again takes the approach of modeling user sessions with Markov Chains. What separates it from the Simple markov model is that it takes the time between consecutive session states into account. The hypothesis of this model is straightforward, a normal user takes some time to submit a query, read the results, hover the mouse to the desired result etc, whereas bots are likely to exhibit excessively frequent behavior.

Appendix A contains Figure 13 which illustrates distribution of time intervals between transition from submit state to other session states. As our hypothesis suggests, most transitions take one or more seconds. However we can also see that the some transitions were faster than one second, which could in some cases

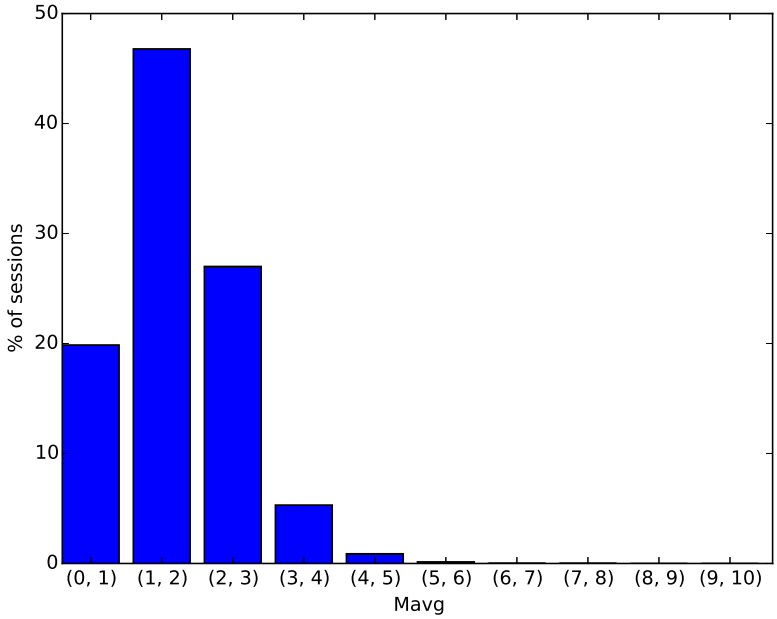


Figure 4: Average Markovian LogLikelihood scores distribution

be interpreted as some sort of miss-click of human users, or as bot behavior in other. To summarize, we could interpret the graphs in various ways, but we can state that time plays a significant role in session state transitions and thus should be incorporated in our model.

6.2.1 Markov Chain states

As already stated at the beginning of the section, this model adds the time information to the simple model. Another new feature is the resubmit state. The resubmit state occurs when a user submits a query that shares at least 50% of tokens of previous query. For instance submitting "news" after previously submitting "news" is a resubmit, "prague airport" after "berlin airport" is also a resubmit.

As a result we model user sessions with session states in form of (State, Interval), the states are defined as follows:

- Submit (S,i): query submit
- Autocompleted Submit (A,i): autocompleted query submit
- Resubmit (W,i): resubmit as defined above
- Next Page Request (N, i): next page request on the SERP page.

- Result Click 1 to 20 (Rj,i): result click as in the simple model
- Other Click (O,i): other click as in the simple model
- IP change (I,i): IP change as in the simple model

In addition to the states, we define the following time intervals as in [10]:

- T0: symbolizes a session end, a session end occurs when the current state is not followed by a new state
- T1: $0 < t \leq 1$ second
- T2: $1 \leq t < 10$ seconds
- T3: $10 \leq t < 30$ seconds
- T4: $t \geq 30$ seconds

In total we distinguish 130 states (26 base states $\times$ 5 intervals) and 130$\times$130=16900 transitions. Note that transitions from states with interval T0 and transitions from IP change to IP change never occur. Another important note is that we did not include more time intervals in order to avoid having a sparse transition matrix.

An example session could be [(A,T2), (W,T3), (O,T3), (R10,T0)]. The interpretation could be that the user performed an autocompleted submit, then read the results for 5 seconds, then resubmitted the query, read results for 20 seconds, then clicked a sponsored click, and after 15 seconds clicked on result with rank of 10.

6.2.2 Transition Matrix

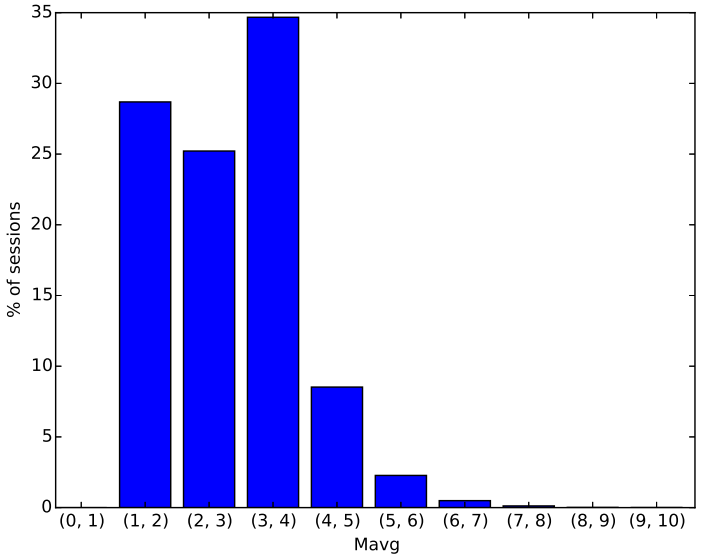
The process of building the transition matrix for this model is exactly as described in the Simple Markov Model section. However, the implementation is slightly different because of the added time intervals and the resubmit state. The extraction is thus implemented in its own Hadoop job called *Time Markovian Model Builder* (more information is provided in the implementation section).

We have extracted a visualization of the transition matrix as in the previous model 14. The visualization shows that the most likely transitions are between states with higher time interval, and thus supports our hypothesis based on expected human user behavior. In addition it also shows that again the most common transitions were between submit (both not and autocompleted) and result click, however this time we can easily see that the time interval between the state change was in range of 1 and 10 seconds.

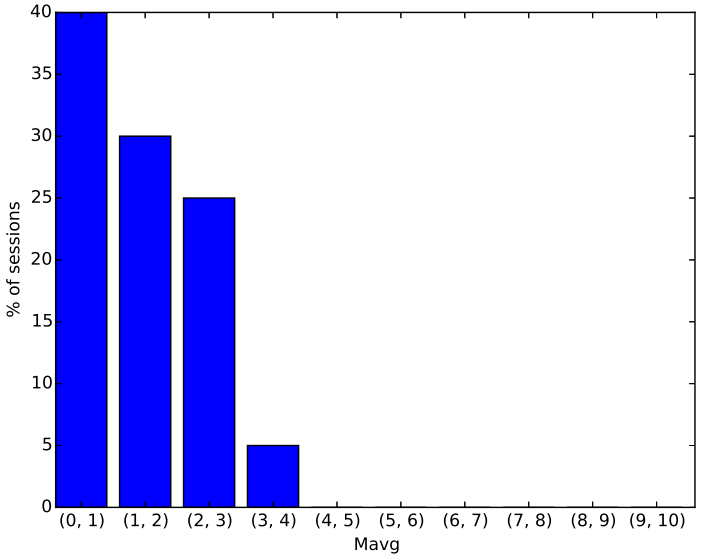
6.2.3 Evaluating the session likelihoods

As with the Simple Markov model, we evaluate the Averaged Maximum Log-Likelihood off all sessions and show the Mavg distribution. The left part of

Figure 5 includes the Mavg distribution obtained by the Time Markov model (and also an example of result of previous attempts).



(a) Mavg scores of Time Markov Model



(b) Mavg scores from [10]

Figure 5: Mavg scores

In contrast with the Mavg distribution obtained by the simple model, the time model does not contain any sessions with likelihood in range (0,1), which is probably caused by the fact that due to the larger number of session states the state transition probabilities are lower in general. Also the first bins are more uniformly distributed and the peak is shifted to the right. To further put the the obtained distribution in perspective, the right part of Figure 5 contains Mavg distribution as shown in [10]:

It can be easily seen that the distributions differ significantly. Unfortunately the authors did not provide further details about the distribution or the transition matrix and therefore we do not know what caused the difference. All we now is that we evaluated more than 400 million sessions compared to 50 million in [10].

6.3 Multiple Feature Model

The Multiple Feature Model provides a lot less compressed view of the user sessions than the previous models. Whereas the former methods used only one single feature (the Average Maximum LogLikelihood) this model uses in total 26 distinct features. The broader view of this model could provide a profitable extension to the previous models. In order to incorporate various user characteristics, the model includes the following feature categories:

- Behavior related features: e.g. JavaScript on/off, night access
- Click related features: e.g. mean time between clicks, clicked domains entropy
- Query related features: average non-alphanumeric characters ratio, URL in query ratio

Some of the features are as proposed in [2], for instance the query keyword entropy or alphabetical ordering. Other features were proposed by the Search engine R&D team, for example the "operators in query" feature or "night access". Other are new e.g. the SERP 2+ clicks.

6.3.1 List of features

This Section includes a list of features which were incorporated in to the Multiple Features model.

We start with the list of user behavior related features:

Javascript Off Whether the user does not interpret JavaScript.Bots often do not interpret JavaScript.

Night access If the user session is during night time. Bots often use the fact that per user request quotas are higher at night.

Operators in query If the user issued a query containing an operator (e.g. -intitle). Bots often use the fact that per user request quotas are higher at night.

SERP 2+ requests Fraction of result page 2 and higher requests to all requests. Human users seldom go to SERP page 2 and higher.

Limit changed If the user requested more results per SERP page. Common users do not known there is such option.

In addition to above listed features, we also use two confidential features provided by the search engine's R&D team.

The list of user click related features follows:

Result click rank entropy The entropy of the ranks of the results clicked by the user. Bots are expected to have a very low or very high click rank entropy.

Clicked domains entropy The entropy of the domains of the results clicked by the user. Bots are expected to have a very low or very high click domain entropy

SERP 2+ click ratio The fraction of clicks on SERP page 2 and higher to all clicks. Human users are expected to click mostly on the first SERP page, whilst some bots might go to very high rank positions.

Mean time between clicks Average time between clicks. Some bots are expected to have a very short time between clicks.

Popular URL clicks Fraction of result clicks on most popular results to all clicks.Human users are expected to click mostly on the popular results.

Trend URL clicks Fraction of result clicks on results whose popularity has risen by more than 50% from last week. Page promoting bots are expected to click on such results.

Result click submit ratio Fraction of result clicks to query submits Some bots click excessively after every submit. Some bots never click.

Other click request ratio Fraction of "other" clicks to page requests. Human users are expected to execute a reasonable number of other clicks.

Click request ratio Fraction of "other" clicks to page requests. Human users are expected to execute a reasonable number of other clicks.

The list of user queries related features follows:

Non alphanumeric ratio The ratio of non-alphanumeric characters submitted to the search engine to all submitted characters. Some bots are expected to submit queries with lots of non-alphanumeric characters (e.g. parts of computer code).

Alphabetical score Alphabetical score as defined in related Section work [2]. Some bots have high alphabetical score.

Average query complexity The average Manhattan distance a single finger has to travel on a keyboard to type the queries submitted by the user. Average human users are expected to have smaller query complexity than some bots.

Query keyword entropy Entropy of the keywords of the queries submitted by the user. Bots are expected to have a very low or very keyword entropy

Query keyword number entropy Entropy of number of keywords in queries submitted by the user. Bots are expected to have a very low or very high number of keywords entropy.

Number of tokens in query entropy Entropy of the number of keywords in queries submitted by the user. Some bots are expected to have a very low number of tokens in query entropy.

Submit time delta entropy Entropy of the the time intervals between consecutive query submits of the user (rounded to minutes). Some bots are expected to issue queries in regular intervals.

Popular tokens ratio Ratio of submits of queries containing popular tokens (e.g. facebook) to all queries. Human users are expected to issue mostly the popular tokens

Trending tokens ratio Ratio of submits of queries containing popular tokens whose popularity has risen by more than 50% from last week. For instance webpage promoter bots are expected to submit mostly such queries.

URL submits ratio Ratio of submits of queries containing an URL to all queries. Some bots are expected to submit excessive number of URL containing queries.

6.3.2 Evaluating user sessions

By evaluating all features of user sessions we obtain a set of vectors with 1×26 dimensions. As in [13] we measure the Mahalanobis distance of all user session vectors to detect outliers (more in Related Work section). In order to compute the Mahalanobis distance, we need to estimate the covariance matrix of the session vector distribution. To do that, we use the maximum likelihood estimate of the covariance matrix of Multivariate Normal Distribution, which is defined as follows:

$$Q = \frac{1}{n} \sum_{i=1}^{n} (x_i - \overline{x})(x_i - \overline{x})^T \tag{17}$$

where x_i is the i-th session vector, n is the number of features (26), and

$$\overline{x} = \begin{bmatrix} \overline{x_1} \\ \overline{x_2} \\ \vdots \\ \overline{x_n} \end{bmatrix} \tag{18}$$

is the sample mean vector. We should point out that such estimation probably causes some error, for the reason that the underlying distribution is most likely multimodal. A reasonable alternative would be to use a Mixture model estimation. Nevertheless, our multiple feature representation of user sessions consumes around 100 GB of storage and thus such estimation would be very demanding. In [12] the users propose the method of density estimation with decision trees, which has a reasonable time complexity and performance and so could be suitable for this task. To use this method is a target for future implementation.

Another important point is that we use Laplace Smoothing to avoid division by zero or zero values when evaluating the fraction (e.g. result click/submit ratio) features:

$$\Theta = \frac{x_i + 1}{N} \tag{19}$$

Where x_i is the original numerator and N is the original denominator.

The current implementation mainly consists of the following Hadoop jobs:

1. RobotFeatureExtractor
2. UserQueryJob
3. UserClickJob
4. SessionDataGrouper
5. AllModelStatsBuilder
6. CovarianceMatrixBuilder
7. Mahalanobis Distance

7 Implementation

In this Section we're going to discuss the core components of our implementation of the proposed solution. Due to the volume of the dataset most of the parts are programmed in Java using the Hadoop framework. Its ability to provide fast parallel read, parallel task execution, process compressed input and produce compressed output are key for our purposes. In addition to Java the Python programming language was used for fast simple statistics extraction and visualization with Matplotlib library [8].

7.1 Simple and Time Markov Model

Even though each of the Markov models use different programs, they implementations share many components, therefore we introduce them together.

7.1.1 MarkovianStateWritable

First common component of the Markov models implementation is the MarkovianStateWritable which implements the Writable interface. Writables are objects used in Hadoop to represent keys or values of the mapreduce programming paradigm. These keys need to support serialization (process of turning structured objects into a bytestream [16]), as they are transferred over network or/and saved to storage between map and reduce tasks.

The MarkovianStateWritables are used to wrap the following data:

- State Code: a number which represents the Markov model states
- Timestamp: is used to indicate the time at which the sessions were in the state defined above
- QueryTokens: contains the query of submit states, is used to detect the resubmit state
- IP: contains the IP address and is used to detect the IP change state

7.1.2 LongPairDoubleWritable

LongPairdDoubleWritable is another Writable object used in our Hadoop jobs. Its main purpose is to represent user/session id in compressed form (as the user id hash is a string representing a 128 bit integer). They include the following data:

- 2 Long values: these values are used to store the hash of session/user ids., which is important for grouping data of the individual users before the reduce phase.
- Timestamp: is being used to sort key, value pairs by time before the reduce phase

The LongPairDoubleWritable also contains a custom compare method, which is used to group session data regardless of the timestamp.

7.1.3 MarkovianModelMapper

Another common component of the implementation of the Markov models is the MarkovianModelMapper, which is used in map task in order to parse the input JSON logs and detect the sessions state data in them, this all happens in parallel. This map task emits pairs of ¡LongPairDoubleWritable, MarkovianStateWritable¿ which primarily represent the triplet ¡Userd, State, Time¿

7.1.4 KeyValue pair partitioning and grouping

In order to be able to group all data related to the individual sessions, we need to implement a custom partitioner, which is a Hadoop framework object that divides the map task outputs to the reducers. This is done in the SessionPartitioner object. By default the Hadoop framework uses the hashcode value of the key to partition data between reducers. however this could lead to data loss in our case, as the keys contain both session id and time data, and we need to group data by session ids regardless of the timestamp.

We also need to implement a custom grouping comparator, which is used to sort our ¡SessionId, Time¿ pairs coming from the map phase according to time. That is implemented in the SessionIdGroupingComparator.

7.1.5 The reduce tasks

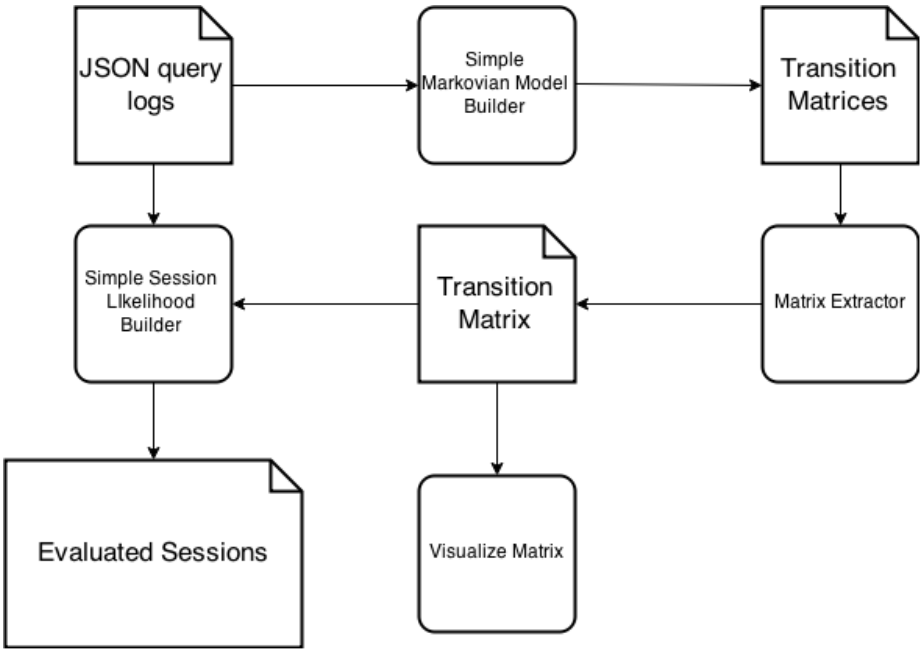
The reduce tasks used in the Markov Model Hadoop jobs are different. However they always process the input triplets of <Userd, State, Time >. In case of ModelBuilder jobs they build the transition matrices (in this case only containing the counts of respective transitions) of the Markov models. In case of SessionLikelihood jobs they evaluate the session likelihoods.

7.1.6 Python scripts

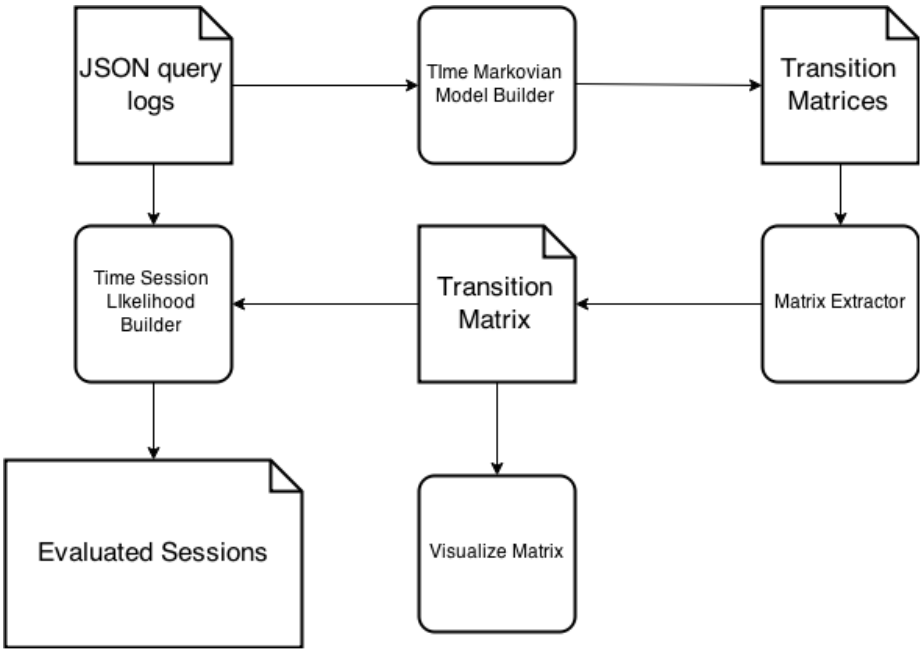
As the ModelBuilder tasks produce around 100 transition matrices, we need to sum them using the Matrix Extractor script to obtain the complete transition matrix containing transition probabilities. In addition we also use the Python script *Matrix Visualization* for visualization.

7.1.7 Summary

We provide a simplified illustration of Simple and Time Markov Models flow charts, which can be seen in Figure 6.



(a) Simple Markov Model Flow Chart



(b) Time Markov Model Flow Chart

Figure 6: Markov models flow charts

7.2 Multiple Features Model

To thoroughly describe all the code related to building the Multiple Features Model would be very lengthy, therefore we will introduce it in brief. To start with we can state that Multiple Features Model uses some of the code introduced in the Markov Model, mainly all the code related to session or user id operations - the LongPairDoubleWritable, SessionIdGroupingComparator and the SessionPartitioner.

Building the Multiple Feature Model consists of the following phases:

- Extraction of query token and result click related data
- Evaluating the session vectors
- Estimating the Covariance Matrix
- Evaluating the Mahalanobis distance

7.2.1 Query and result click data extraction

In order to be able to evaluate the query and result click related features, we need to extract the trend data. The trend data is being extracted from the JSON query logs by the KeywordVocabularyBuilder and ResultsVocabularyBuilder jobs. These jobs output a list of in the following form:

```
query_token [true/false true/false ... i]
```

Where each field i of the boolean array indicates if the results/tokens popularity has grown in week i by more than 50% from previous week. The first element is true if the results popularity in first week is more than 150% of the mean popularity.

This list of queries is also used in the TopTokensBuilder job, which creates a list of most popular results and query tokens.

7.2.2 Extraction of the session vectors

Once we have the trend and popularity data we can proceed to extraction of the session vectors from the query logs. The extraction is implemented in the following jobs:

- RobotFeatureExtractor: extracts the behavioral features of sessions
- UserQueryJob: extracts the query related features of sessions
- UserClickJob: extracts the click related features of sessions
- AllModelBuilder: can be used as an alternative to previous jobs, as it extracts all needed features and builds the session vectors at once

When the features are extracted, the outputs are sent to the SessionDataGrouper (not needed in case of use of the AllModelBuilder), which groups the distinct features in single vectors. This produces around 100GB of data.

7.2.3 Covariance Matrix Estimation

Before we can estimate the Covariance Matrix, we need to obtain the mean vector from the session vectors. This could be done by a simple python script, however copying tens of gigabytes of data from HDFS is lengthy and thus we use the AllStatsBuilder job which extracts the mean vector (and other statistics) in a few seconds. The mean vector and the session vectors are then sent to the CovarianceMatrixEstimate job, which outputs the covariance matrix estimate.

7.2.4 Evaluating the Mahalanobis Distance

Once we have extracted the covariance matrix, we can run the MahDistance job, which evaluates the Mahalanobis distance of the session vectors. A shortened flow chart of the Multiple feature model can be seen in Figure 7.

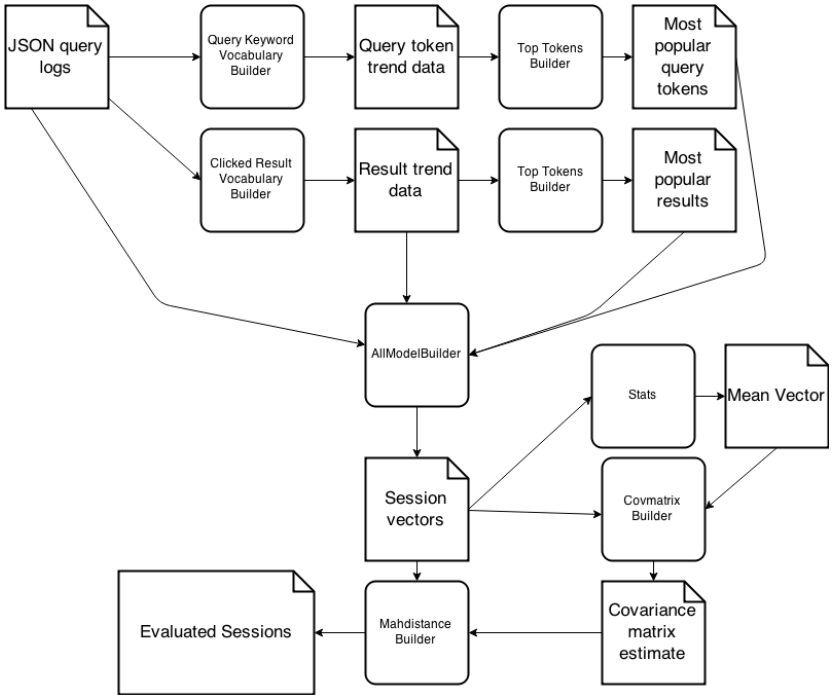


Figure 7: Multiple Feature Model shortened flow chart

8 Evaluation

As we have already stated in Section 5, we do not have access to a labeled dataset and thus evaluation of the proposed models is complicated. Most of previous attempts named in Section 3 relied on manual evaluation. However manual evaluation can evaluate only a small fraction of session data and human "judges" can be biased, therefore we have decided to use different methods. Nevertheless the search engine company's R&D team provided us with a suggestion of how to identify human or bot users with help of 4 binary features. Unfortunately the features are confidential and therefore we can't provide their detailed description. A general description of the suggested features follows:

1. The first feature can be used to identify users who are "most likely human". The feature indicates that the user uses some of the other services provided by the search engine company.
2. If the second feature is present in the user session, it implies that the user is "most likely bot". This feature is related to how the user submits the queries to the search engine.
3. The third feature also helps to identify "most likely bot" users. It is related to how the user's internet browser (or script) interprets the search engines web page.
4. The last feature is related to how many results per SERP page the user requests and is also used to detect users who are "most likely bot".

In the following text we will refer to the "most likely human" feature simply as *human feature* and all of the "most likely bot" *bot feature*.

8.1 Evaluation Stages

In order to provide a more profound evaluation and analysis of the proposed models, our evaluation consists of the following stages (more on the stages in their respective sections):

- Visual Evaluation: its purpose is to provide a visual analysis of the models outputs
- Metrics Evaluation: evaluation using standard metrics
- Automated Evaluation: tests based on multiple bot behavior scenarios

8.2 A fourth model

In addition to evaluating the base three models, we have decided to evaluate one additional model, which incorporates the likelihood obtained by Time Markov Model (TMM) in the *session vectors* of the Multiple Feature Model (MFM). The decision to do so has been made after analyzing the correlation between the TMM and MFM models in Visual Evaluation section 8.4. For the purpose of the whole evaluation section, we refer to this model as Mixed Model (MM).

8.3 Preprocessing

The preprocessing phase of evaluation consisted of the following subtasks:

1. Training the models: Building the transitions matrices of the Markov Models
2. Filtering out the *short sessions*
3. Evaluation of all Maximum Likelihoods of Markov Models, evaluation of Mahalanobis distance of Multiple Feature Model
4. The evaluation itself

Note that we did not create any data splits (train/test) due to the following reasons:

- Not putting aside any data for testing will leave us more data for fitting a better density estimator. In our scenario, where we do not use any (hyper)parameter that would need to be tuned to make a better fit, the use of validation (test) set would be pretty much for nothing.
- It is true that the density estimator (Markov chain model/Gaussian distribution model) is subsequently turned into a classifier by introducing a threshold (for likelihood/distance), but that was not the primary goal of learning. And because the threshold for classification is selected from a finite set, which can be very small (for example, it equals to the number of bins for Markov chain model likelihoods +1) but even in the extreme case of one unique threshold per session, with our amount of data the observed (empirical) performance of the classifier should match its expected performance.
- The model is being learned using unsupervised methods, regardless of the classification error. Thus the resulting evaluation of the quality is very similar to evaluation on test dataset.

8.4 Visual Evaluation and Analysis

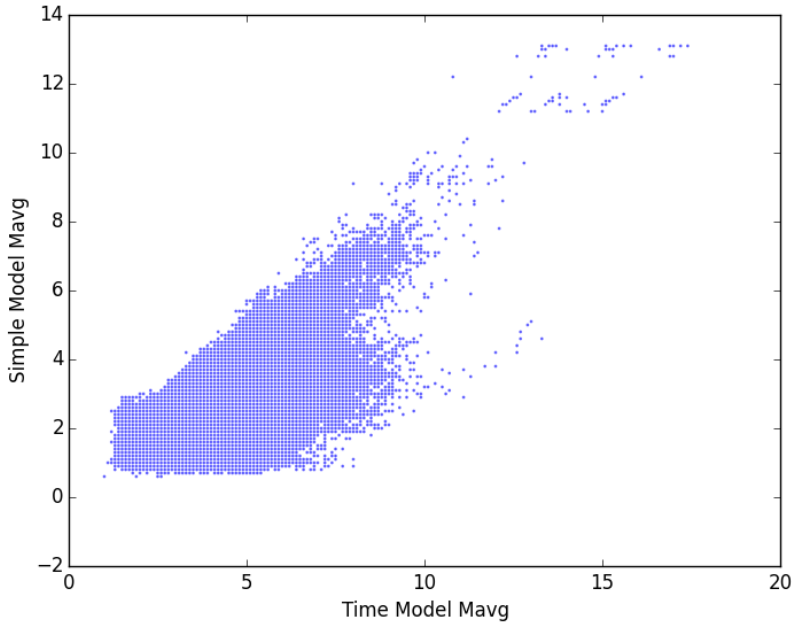
Visualizations are a useful extension to the evaluation with standard metrics, which enables us to further analyze the outputs of our models. Figures 9, 10 and 11 contain the following plots:

- A histogram of the output distributions of the respective models
- ROC curve plot: a plot of recall against the false positive rate F ($F = \frac{FP}{FP+FN}$), which allows us to see the "accuracy" of the model
- Human/Bot Feature Distribution: a histogram containing the distribution of the human/bot feature against the bins. Thus the height of the bar above a bin indicates how many % of all users with given feature belong to

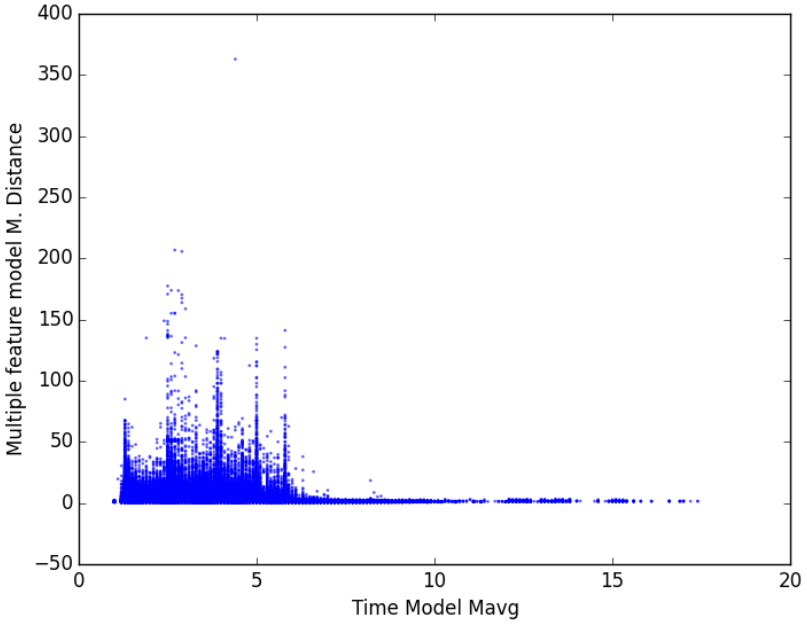
the bin. This plot also contains a not-in-scale histogram of the respective output distribution for comparison. The more a robot feature distribution is "positioned" in the longtail of the output distribution, the more we can expect the model output to be relevant for classification. On the other hand if a human feature distribution is "positioned" mostly in the longtail, the classification accuracy is likely to be low.

In addition to above mentioned plots, the Figure 8 contains two additional (scatter) plots. The left scatter plot (a) enables us to analyze the correlation between the two Markov models. It can be seen that the models are slightly positively correlated as most points lie around one diagonal line. Therefore we can state, that although the Time Model is more complex and incorporates more information, in the end it provides very similar results.

The scatter plot (b) compares the MFM's Mahalanobis Distance distribution against the TMM's distribution. It can be seen that the models are rather uncorrelated. That means that they provide different output information, which could be complementary for each other, however it could also cause ambiguous results when using both models for classification. The "most ambiguous" part of the plot is the lower right area, where the M. distance is just above zero, whilst the Mavg shows very high values over 10. After a deeper inspection, we have found out that this region contains 612 sessions, which is only a very small fraction of all 400 million sessions. However, 32% of the sessions contained one of the bot features, 28% both human and bot features, which makes 60% of potentially atypical sessions and represents a possible blind spot of the MFM. Therefore we have decided to create one additional model for evaluation, which is based on the MFM, but adds the TMM likelihood as an additional element of the *session vectors*. Such model could theoretically have an advantage in the above mentioned ambiguous cases (over the separated models). Note that the authors of [13] have taken a similar approach, more in Related Work section 3.2.



(a) Time and Simple Model scatter plot



(b) Time and Multiple Features model scatter plot

Figure 8: Scatter plots

8.4.1 Results

We have already discussed the provided scatter plots, now we provide an analysis of the remaining plots of all three models.

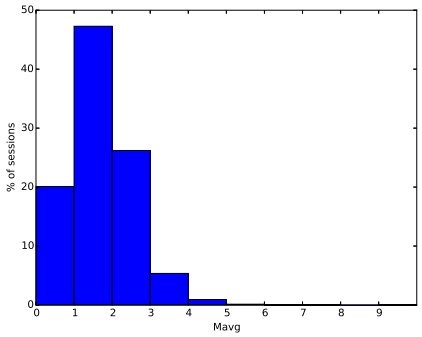
Firstly, we discuss the Simple Markov Model(SMM) and Time Markov Models (TMM). As their corresponding plots in Figures 9 and 10 contain very similar patterns, we have decided to provide only one commentary for both models.

The distributions were discussed in Section 6. However we can use them to help analyze the human/bot features distributions below it.

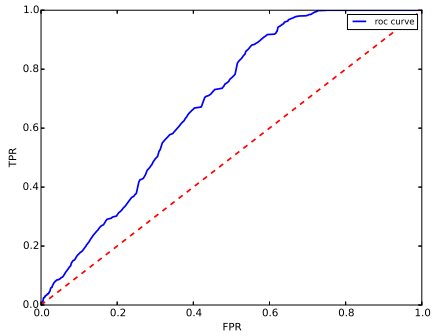
The ROC curves of both models show, that their classification accuracy on the set of provided features is rather poor, as it lies close to the dashed diagonal line.

Human feature distributions of the respective models are both positioned all across the output distributions. Which indicates a high possible false positive ratio if the models are used for classification.

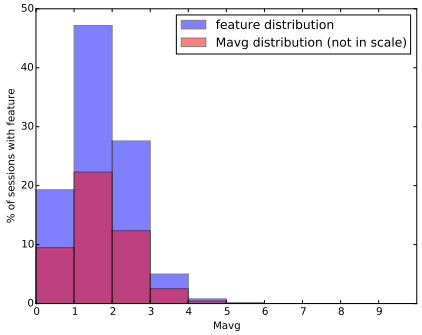
Bot features distributions are similar to the output distribution which complicates selection of potential classification thresholds and thus makes reasonable classification almost impossible.



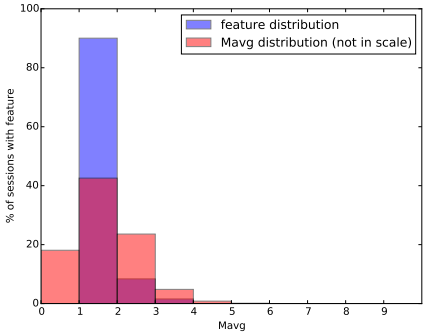
(a) SMM Mavg Distribution



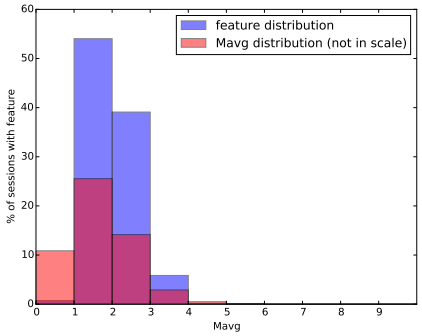
(b) SMM ROC curve



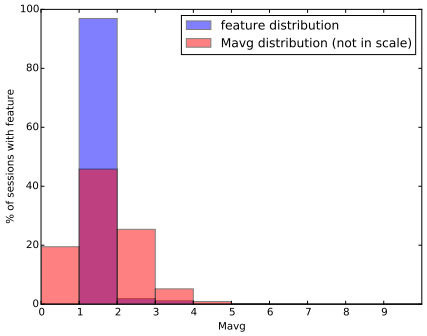
(c) Human indicating feature distribution



(d) First bot indicating feature distribution

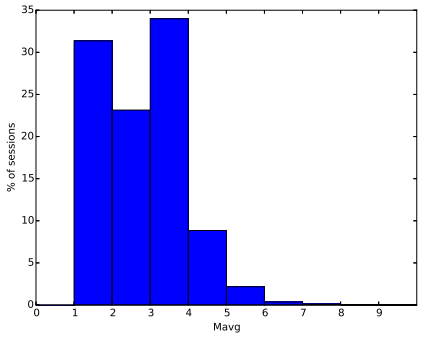


(e) Second bot indicating feature distribution

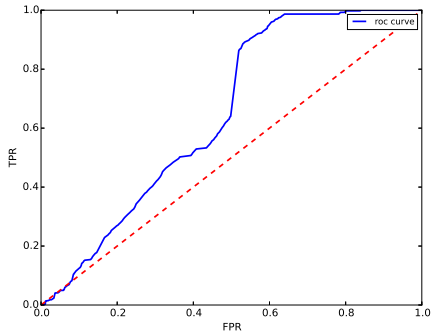


(f) Third bot indicating feature distribution

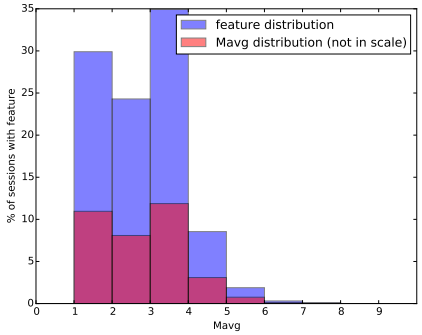
Figure 9: SMM evaluation



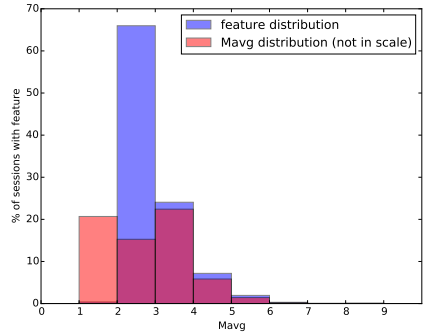
(a) TMM Mavg Distribution



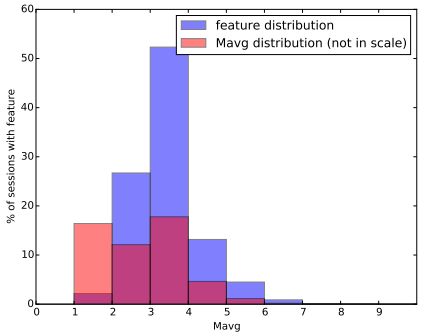
(b) TMM ROC curve



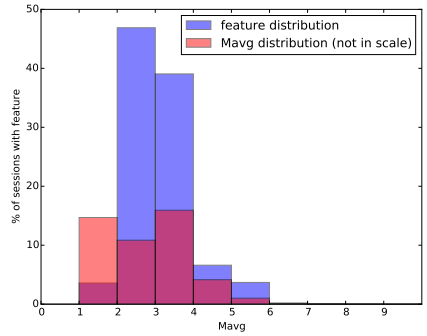
(c) Human indicating feature distribution



(d) First bot indicating feature distribution



(e) Second bot indicating feature distribution



(f) Third bot indicating feature distribution

Figure 10: TMM evaluation

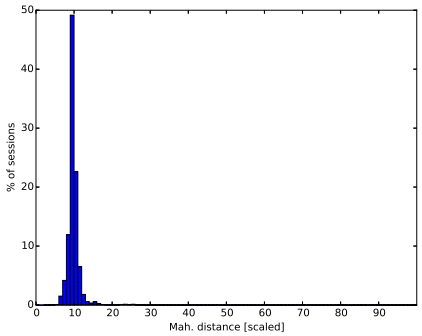
Secondly, we discuss the Multiple Features Model (MFM) whose respective plots can be seen in Figure 11.

The distribution is different from the Markov Models as it contains a high peak and a long tail.

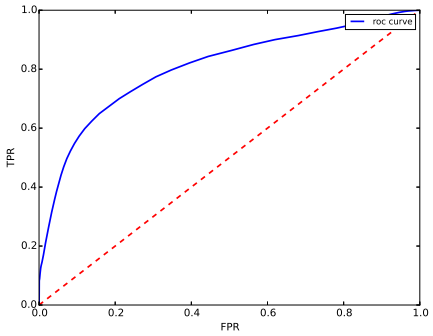
The ROC curve's shape is considerably more promising for classification than those of the Markov Models as it lies further from the dashed diagonal, which means a better bot/human classes separation performance.

Human feature distribution mimics the output distribution. Which again complicates potential classification.

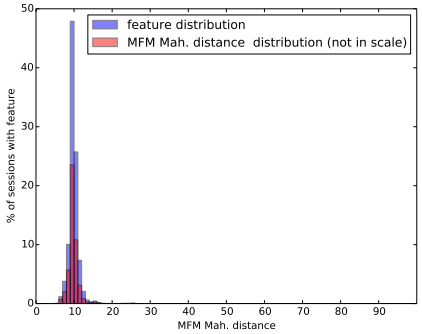
The first and third Bot features distributions are placed only in the long tail, which is a good sign for possibly high true positive ratio. Unfortunately most of the second feature's "weight" overlaps the output distribution, which will increase the false negative ratio. However, this is the third time the second bot feature mimics the output distribution. We do not know whether this indicates a blind spot of our models, or the bot feature is not robust enough for evaluation purposes. In the Evaluation using Metrics section 8.5, we show that if we ignore this feature during evaluation, the metrics scores obtained by the MFM and the MM rise significantly.



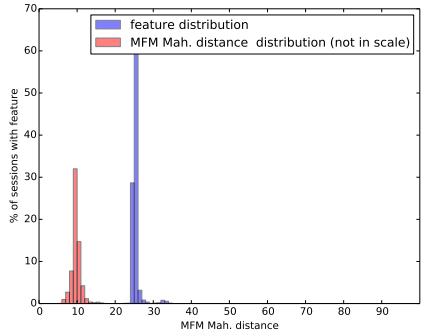
(a) MFM Mahalanobis Distance Distribution



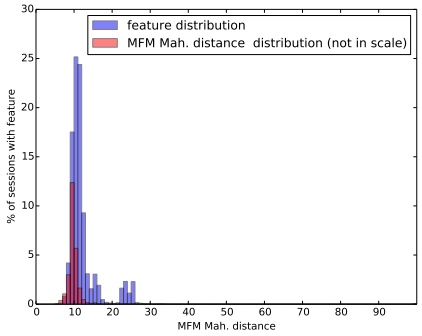
(b) MFM ROC curve



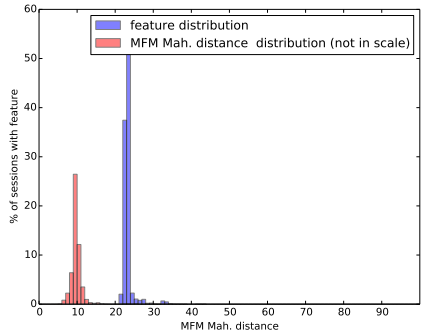
(c) Human indicating feature distribution



(d) First bot indicating feature distribution



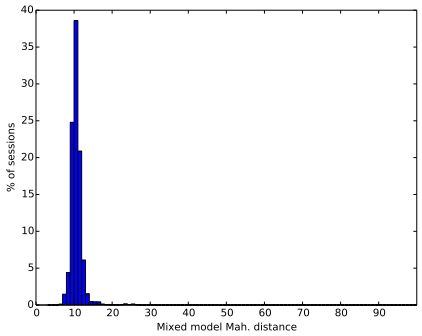
(e) Second bot indicating feature distribution



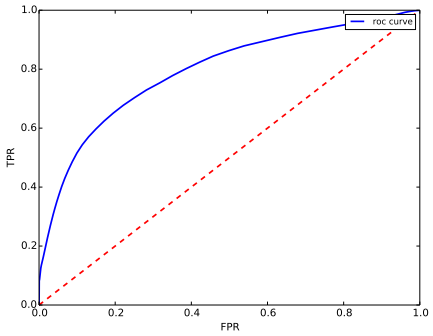
(f) Third bot indicating feature distribution

Figure 11: MFM evaluation

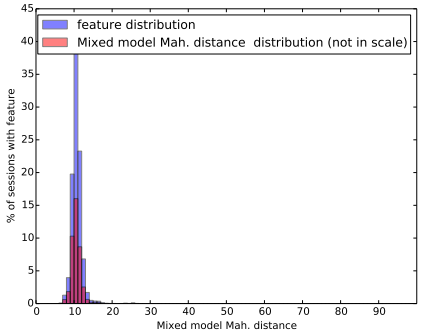
Thirdly, we provide a visual analysis of the MM. However, it can be easily seen that the model's distribution, ROC curve and all bot and human features distributions are very similar to those of MFM's. This implies that the addition of the session likelihoods to the MM did not provide a significant amount of new information over the MFM.



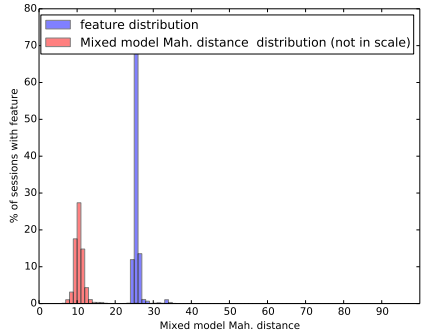
(a) MM Mahalanobis Distance Distribution



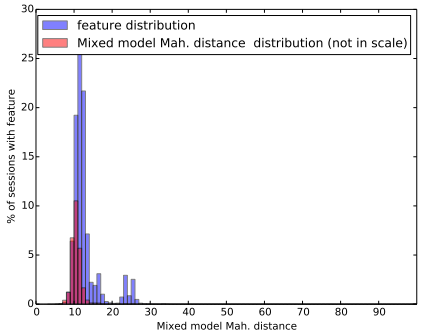
(b) MM ROC curve



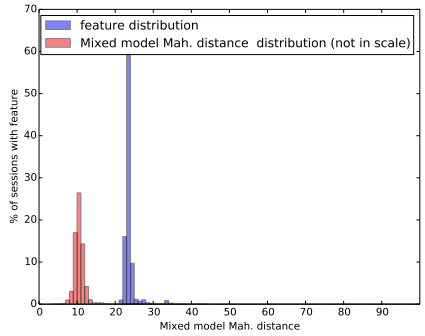
(c) Human indicating feature distribution



(d) First bot indicating feature distribution



(e) Second bot indicating feature distribution



(f) Third bot indicating feature distribution

Figure 12: MM evaluation

8.5 Evaluation using Metrics

The following list contains all metrics that will be used throughout the evaluation section. Note that TP stands for true positives, (TN) for true negatives, (FP) for false positives and finally (FN) for false negatives (all of them are described below the list).

- Precision: this metric is defined as $P = \frac{TP}{TP+FP}$, where P is the Precision
- Recall: this metric is defined as $R = \frac{TP}{TP+FN}$, where R is the Recall
- Accuracy: this metric is defined as $A = \frac{TP+TN}{TP+FP+TN+FN}$

Let's now define the true/false positive/negative cases:

- TP: true positive case, a session was marked as a bot session, and at least one of bot features was present in the session.
- FP: false positive case, a session was marked as bot session whilst no bot features were present in the session.
- TN: true negative case, a session was labeled as a human session, no bot feature was present in the session.
- FN: false negative case, a human labeled session contained one of the bot features.

Note that all the cases above depend only on bot features, which means that we expect a session with any bot feature to be a "bot" session, regardless of if a human feature is present in the session or not. On the other hand, we expect all sessions without any bot feature to be "human".

8.5.1 Results

Table 1 contains the final evaluation results of all models. It can be easily seen that the MMF's and MM's performances were by far the best, achieving three times better precision and recall than the Time Markov Model and six times better than the Simple Model. Both Markov models have thus proven that such simple (although natural) representation of sessions is not sufficient for classification.

To compare the multiple feature models, we can state that the MFM has a slightly better precision and recall than the MM. On the other hand, the MM achieved a slightly better accuracy score. However, as has already been stated in Visual Evaluation section8.4, it is clear that the added session likelihood feature does not enable the MM to provide significantly different outputs or better performance.

Model	Threshold	Precision	Recall	Accuracy	Sessions detected	%Sessions detected
Simple Markov Model	3.2	5%	5%	90%	19134927	4.7%
Time Markov Model	4.51	9%	9%	90%	23775684	5.9%
Multiple Features Model	11.7	34%	34%	92%	21238142	5%
Mixed Model	12.6	32%	31%	93%	20493888	4.8%

Table 1: Models Metric Evaluation

According to the scores obtained by the models, their current performance can't be considered (even remotely) sufficient for real-world deployment. However, as we have suggested in the Visual Evaluation, we have decided to add another table (2), which includes the scores obtained by the models when ignoring the second bot feature, whose distribution overlaps the output distribution of all models (as of now, we do not know if it indicates a blind spot of our models, or the bot feature is not robust enough for evaluation purposes). Note that this leaves us with two bot features for evaluation.

Model	Threshold	Precision	Recall	Accuracy	Sessions detected	%Sessions detected
Simple Markov Model	6.1	2.1%	2.2%	99.1%	1856634	4.6%
Time Markov Model	5.3	0.6%	2%	97%	7187292	17.9%
Multiple Features Model	11.7	86%	86%	99.8%	1718061	4.3%
Mixed Model	22.3	86%	86%	99.8%	1624783	4.1%

Table 2: Models Metric Evaluation, second bot feature ignored

The results of the SMM and TMM have both worsened. On the other hand, the scores of the MFM and MM models have increased dramatically. To sum up, the performance of MFM and MM models is promising according to the

first evaluation, and even more according to the second. In addition, we must state that the MM and MFM models can be relatively easily improved with further work, for instance, by implementing better session vector space density estimation. Thus they could achieve even better results.

8.6 Automated Evaluation

Another extension to the evaluation process is automated evaluation. We have prepared multiple bot scenarios to asses the the ability to detect some of the common atypical behavior.

8.6.1 Testing scenarios

We have prepared the following bot scenarios:

- Rank check bot, Scraper: this bot only submits queries and clicks only on the "next page" button
- Crawler, Academical Bot: submits queries and clicks on all results
- Result Promoter: clicks only on results linking to its target domain
- Deplete Competitions Advertising Budget Bot: submits queries and clicks only on sponsored results

the aforementioned scenarios can have (among many other) the following configurations:

- Number of request in session
- Mean time interval between clicks
- Random or repeating interval between clicks
- Bot only or human simulating behavior
- Bot to Human requests ratio
- Target queries vocabulary size
- Bot changes/keeps IP address
- URLs in query ratio

We must acknowledge that relevance of such "white box" testing approach is questionable as the scenarios mostly represent a rather "extreme behavior" and never represent a "normal" use. However we were not provided with any bot sample for "black box" testing and thus we had to create our own implementation.

8.6.2 Results

The results of automated evaluation can be seen in Table 3. Note that the classification thresholds were chosen from Figures 9, 10 and 11, so that they represent a value at which the distribution of the respective model drops significantly.

Model	Threshold	bot types tested	% bots detected	% notdetected	ranking
Simple Markov Model	4	684518400	5%	95%	3
Time Markov Model	5	684518400	43%	57%	1
Multiple Features Model	15	684518400	42%	58%	2

Table 3: Automated Evaluation Results

The table shows that the Time Markov Model provided the best detection performance, whilst the Simple Markov Model the worst. This could have been caused by the fact that (as mentioned before) the Simple Markov Model does not take time between session states into account, considering the fact that many of the bot configurations contained rapid state transitions (under 30 seconds intervals). The Multiple Features Model's performance was on par with the Time Model, its advantage over the Simple Model could have been that it incorporates many click and query related features. Note that we did not include the MM to the automated evaluation procedure, as we were rather interested in its metric scores and visualizations.

8.7 Summary

As we did not have access to any labeled data for evaluation, we had to rely on the following evaluation methods:

- Metrics Evaluation using 3 *bot* features
- Visual Evaluation using plots of distributions, ROC Curves, scatter plots
- Automated Evaluation based on multiple bot behavior scenarios

Even though our testing methods were (due to absence of ground truth) considerably theoretical, we can state the following

- Firstly, the performance of the Markov models has shown that such simple although natural representation is not suitable for atypical user behavior

detection task even though previous research (whose evaluation mostly relied on human annotation) states otherwise.

- Secondly, the MFM exhibited a much better performance. But still it is not sufficient for real world use. However its results can be considered promising, and the model might be relatively easily improved, for instance by implementing better session vector space density estimation or addition/removal of features. This will most likely be the task for future work.
- Thirdly, as the visualizations show, one of the bot features overlaps the output distributions of all proposed models. We do not know whether this indicates a blind spot of our models, or the bot feature is not robust enough for evaluation purposes. However, when ignoring this feature, the MFM achieved precision of 86%, recall 86% and accuracy 99%, while identifying 18 millions of sessions as bot session.
- We have evaluated a fourth model, which adds the session likelihood obtained by TMM to the MFM *session vectors*. However, this model delivered more or less the same performance as MFM.

9 Conclusion

Our task was to design and implement a statistical model for detecting atypical users of a search engine. After a thorough analysis of the current state of the art, we have implemented three unsupervised detection models, which all incorporate some of the fundamental findings of previous works.

The first two models focus on interpreting user sessions as Markov Chains, where the Markov Chain states are represented by various user actions (submits, clicks etc.), one of the models also uses time intervals between state transitions to distinguish between multiple substates. Although the overall atypical user detection performance (recall and precision) of these models is rather poor, they help to provide insight into various user behavior characteristics, for instance, the difference between common and rare clicking behavior.

The third model provides a broader view on user sessions and user behavior in general. Its method consists of two phases. During the first phase it transforms user sessions into a *session vectors* of 26 features, which characterize the users behavior, clicking patterns and query patterns. The second phase consists of estimating the session vectors distribution and computing the Mahalanobis distance of the session vectors.

In the absence of ground truth data, we have built a three stage tentative evaluation method consisting of automated testing, visualization and detection of features suggested by the search engine company's R&D team.

In the three stage evaluation procedure the *session vector* model delivered a promising performance whilst achieving accuracy of up to 99% and detecting 18 millions of sessions as atypical, with precision and recall from 34% to 86%, depending on the evaluation setup. Nevertheless, we must emphasize, that without annotated data or more robust evaluation criterions, the results are only preliminary.

Future work will focus, among other things, on improvement of the estimation of the underlying *session vector space* distribution and addition/replacement of features. However, a key part for building a complete atypical user detection model is to have a labeled dataset for evaluation. In case of closer cooperation with the search engine, we could mine more suspicious user activity data, which would significantly help further development of the detection model. This could be achieved by using, for example, a proxy servers list, Tor anonymity network gates list or by reverse engineering of the commercially available search engine scrapers.

References

[1] Cloudera Developer Blog. How-to: Select the right hardware for your new hadoop cluster. `http://blog.cloudera.com/blog/2013/08/how-to-select-the-right-hardware-for-your-new-hadoop-cluster/`, 2015. Accessed: 2015-05-21.

[2] Greg Buehrer, Jack W Stokes, and Kumar Chellapilla. A large-scale study of automated web search traffic. In *Proceedings of the 4th international workshop on Adversarial information retrieval on the web*, pages 1–8. ACM, 2008.

[3] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. An experimental comparison of click position-bias models. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 87–94. ACM, 2008.

[4] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. John Wiley & Sons, 2012.

[5] Omer M Duskin and Dror G Feitelson. Distinguishing humans from bots in web search logs.

[6] The Apache Software Foundation. Apache hadoop, 2015. [ONLINE; accessed 2015-04-21].

[7] Fan Guo, Chao Liu, Anitha Kannan, Tom Minka, Michael Taylor, Yi-Min Wang, and Christos Faloutsos. Click chain model in web search. In *Proceedings of the 18th international conference on World wide web*, pages 11–20. ACM, 2009.

[8] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[9] Hongwen Kang, Kuansan Wang, David Soukal, Fritz Behr, and Zijian Zheng. Large-scale bot detection for search engines. In *Proceedings of the 19th international conference on World wide web*, pages 501–510. ACM, 2010.

[10] Xin Li, Min Zhang, Yiqun Liu, Shaoping Ma, Yijiang Jin, and Liyun Ru. Search engine click spam detection based on bipartite graph propagation. In *Proceedings of the 7th ACM international conference on Web search and data mining*, pages 93–102. ACM, 2014.

[11] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. 1999.

[12] Parikshit Ram and Alexander G Gray. Density estimation trees. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 627–635. ACM, 2011.

[13] Narayanan Sadagopan and Jie Li. Characterizing typical and atypical user sessions in clickstreams. In *Proceedings of the 17th international conference on World Wide Web*, pages 885–894. ACM, 2008.

[14] StackExchange. Calculating log-likelihood for given mle (markov chains). `http://stats.stackexchange.com/questions/47685/calculating-log-likelihood-for-given-mle-markov-chains`, 2014. Accessed: 2015-05-21.

[15] Eric Wesstein. L2-norm from wolfram mathworld. `http://mathworld.wolfram.com/L2-Norm.html`, 2015. Accessed: 2015-05-21.

[16] T. White. *Hadoop: The Definitive Guide: The Definitive Guide*. O'Reilly Media, 2009.

[17] Kenneth C Wilbur and Yi Zhu. Click fraud. *Marketing Science*, 28(2):293–308, 2009.

[18] Fang Yu, Yinglian Xie, and Qifa Ke. Sbotminer: large scale search bot detection. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 421–430. ACM, 2010.

[19] CESNET z. s. p. o. Hadoop. `https://www.metacentrum.cz/cs/hadoop/index.html`, 2015. Accessed: 2015-05-21.

10 Appendix

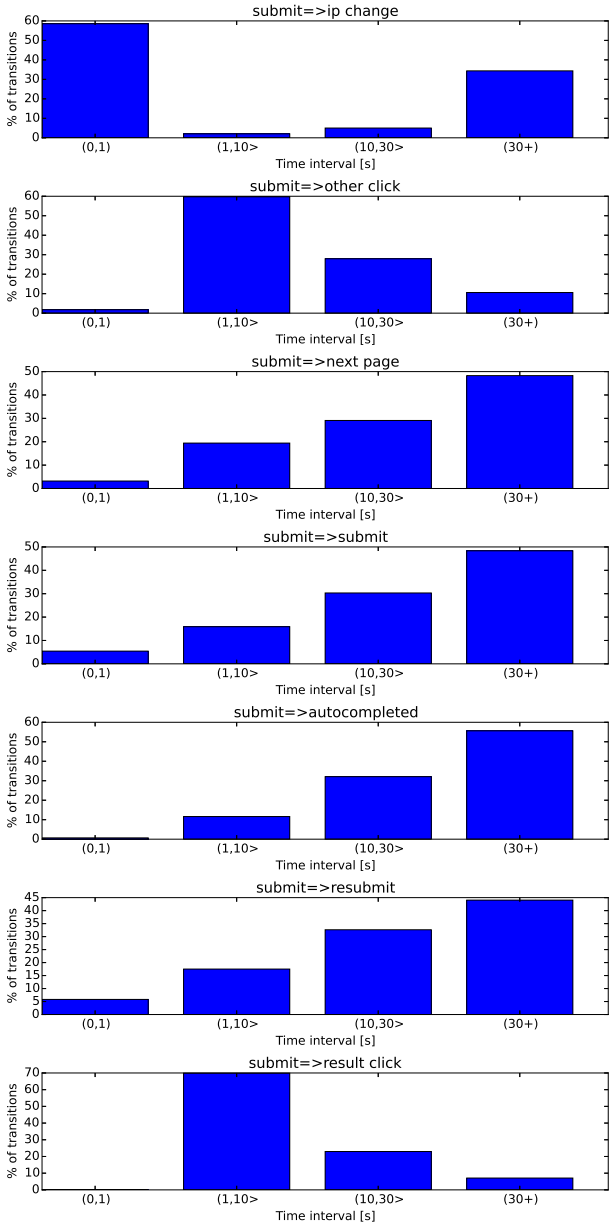


Figure 13: Submit to other state time interval distributions

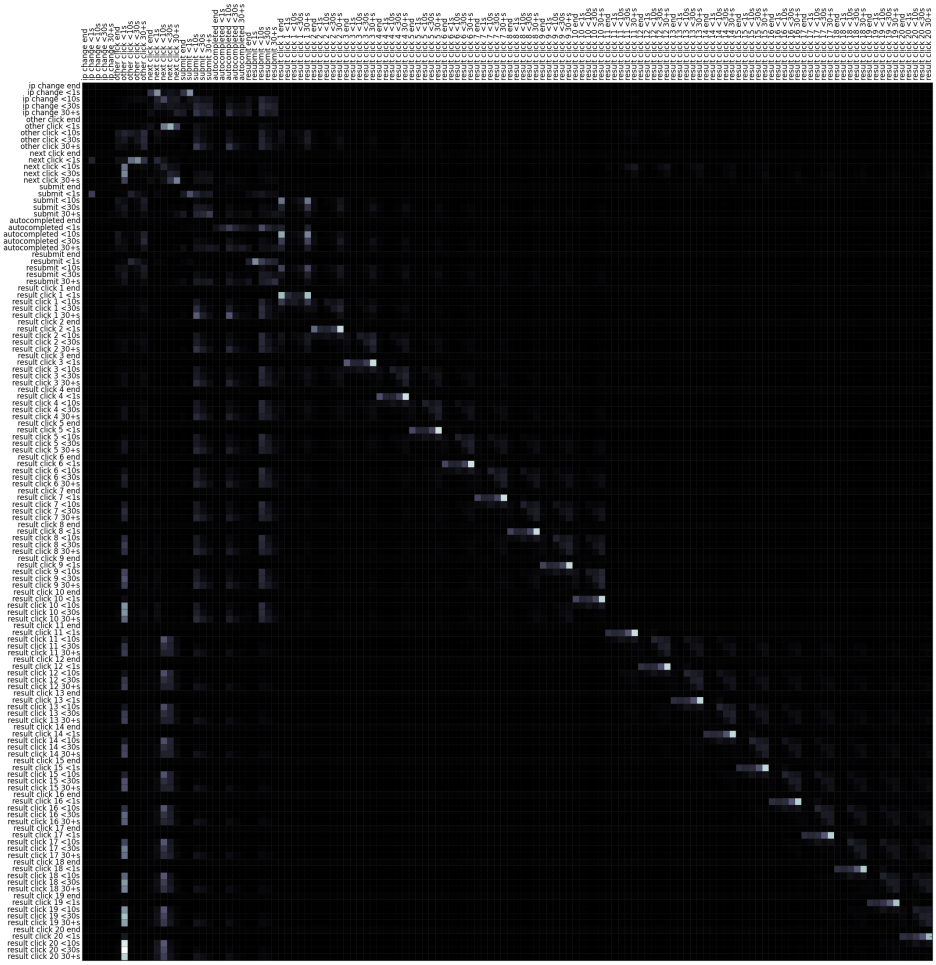


Figure 14: Time Markov Model Transition Matrix