A Supervised Topic Transition Model for Detecting Malicious System Call Sequences

Han Xiao
Technische Universität München
Fakultät für Informatik
Boltzmannstraße 3
85748 Garching, Germany
xiaoh@in.tum.de

Thomas Stibor
Technische Universität München
Fakultät für Informatik
Boltzmannstraße 3
85748 Garching, Germany
stibor@in.tum.de

ABSTRACT

We propose a probabilistic model for behavior-based malware detection that jointly models sequential data and class labels. Given labeled sequences (harmless/malicious), our goal is to reveal behavior patterns and exploit them to predict class labels of unknown sequences. The proposed model is a novel extension of supervised latent Dirichlet allocation with an estimation algorithm that alternates between Gibbs sampling and gradient descent. Experiments on real-world data set show that our model can learn meaningful patterns, and provides competitive performance on the malware detection task. Moreover, we parallelize the training algorithm and demonstrate scalability with varying numbers of processors.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Learning—*Parameter learning*; I.5 [**Pattern Recognition**]: Models—*statistical*

General Terms

Algorithms, Experimentation, Security

Keywords

Probabilistic Model, Supervised Learning, Sequential Data, Malware Detection

1. INTRODUCTION

Detecting malware, that is, malicious programs such as Trojan horses and worms is an active field of research in computer security [10]. One approach is to monitor generated system call sequences of the observed programs and to apply machine learning techniques to classify such sequential data [9, 11]. A system call sequence can have for instance the following form: "`OpenRegistry, ManipulateRegistry, OpenSocket, WriteSocket,...`" and characterizes a *malicious* program that performs manipulations in the Windows registry database and transmits information by means of network socket operations. From a machine learning perspective, two types of information are encoded in a system call sequence, namely, semantic and sequential information. Semantic information denotes system calls such as "`ReadFile, WriteFile`" and "`CloseFile`" that jointly belong to a single semantic topic, here, file I/O operation. In contrast, system calls such as "`Connect, Listen`" and "`SendBuf`" are tightly coupled to the semantic topic network communication. In other words, any system call sequence can be characterized by a set of latent topics. Such an assumption is also suggested in the field of information retrieval [8] and text analysis [2], where it is assumed that text documents are generated by a random mixture of latent topics [13]. Sequential information denotes the Markovian dependence of system calls, that is, dependence of the next system call given the preceding system calls. In summary, semantic and sequential information encoded in system calls carry the succinct behavior of a program, thus they are crucial information to be exploited for detecting malware.

Inspired by recent machine learning research in probabilistic topic models, we focus on learning patterns in sequences and predicting labels of *unseen* system call sequences. Discovering patterns via topic models has been extensively studied in the literatures. First, latent Dirichlet allocation (LDA) [2] has been successfully employed to discover contextual information in data. Next, Simplicial Mixture of Markov Chain [4] and the Topic Model [14] are proposed. These approaches directly model the Markovian dependence of the conditional probability of a symbol given its previous state. Additional approaches where the Markovian property is integrated into topic models are proposed in [5, 6]. Moreover, to discover topics as well as phrases, i.e. the local dependency between words, Topical N-grams [15] are proposed. All of the above models enjoy a big success in text modeling.

Our proposed model finds a set of topics that are representative of both behavior patterns and class labels. The two main contributions of this work are:

1. The supervised latent Dirichlet allocation (sLDA) [1] is modified and extended to fit our problem domain. That is, the proposed model provides a multi-class extension of sLDA for predicting discrete response values, via generalized logistic regression and can be trained with a straightforward parallized algorithm. Moreover,

Markovian dependence is integrated to model the sequential nature of the system call data.

2. We classify malware and extract behavior patterns in a *single* model. Previous approaches usually performed these two tasks separately by treating them as different parts of a pipeline. That is, first selecting features and then feeding features to a classifier. On contrary, we fundamentally take the probabilistic approach to solve classification and pattern analysis simultaneously.

We explain the proposed model and present the estimation algorithm in Sect. 2. Experiments on the real-world data set are presented in Sect. 3. Sect. 4 concludes. For the sake of conformity to terms used in the field of probabilistic topic models, the following name convention is used throughout this paper. A system call is termed *word*. A system call sequence generated by a program is termed *document*. A set of system call sequences is denoted as a *collection*.

2. SUPERVISED TOPIC TRANSITION MODEL

The notation used in this paper is summarized in Fig. 1. The characteristic of our proposed *Supervised Topic Transition* model (STT) is threefold. First, we add a transition matrix between different topics in each document in order to find the sequential correlation between topics. As in the LDA model, topic-words distributions are shared by all documents. Second, we employ generalized logistic function for incorporating the multi-class labels to the model, thus provides an oracle on exploring latent space meanwhile gives the model discriminative power. Third, we implement the training algorithm for the STT model in a parallel manner. This allows our model to be highly efficient on large-scale sequential data. The generative process of the STT model can be described as follows:

1. Draw multinomial distributions ψ_z from a Dirichlet prior β for each topic z;

2. For each document d, draw T multinomial distributions $\phi_{d,z}$ from a Dirichlet prior α; then for each word $w_{d,i}$ in document d:

   (a) Draw $z_{d,i}$ from multinomial $\phi_{d,z_{d,i-1}}$

   (b) Draw $w_{d,i}$ from multinomial $\psi_{z_{d,i}}$

3. Draw a class label y_d for document d from a generalized logistic function $P(y|\bar{n},\theta)$, where $\bar{n}_{d,z,z'} = n_{d,z,z'}/\sum_{z'=1}^{T} n_{d,z,z'}$ is the empirical topic transition frequencies. The generalized logistic function provides the following distribution:

$$P(y_d = k|\bar{z},\theta) = \frac{\exp(\sum_{z,z'}\theta_{k,z,z'}\bar{n}_{d,z,z'})}{\sum_{k=1}^{K}\exp(\sum_{z,z'}\theta_{k,z,z'}\bar{n}_{d,z,z'})}. \quad (1)$$

Symbol	Description
T	number of topics
D	number of documents
V	number of unique words
K	number of classes
N_d	number of words in document d
$w_{d,i}$	the ith word in document d
$z_{d,i}$	the topic associated with the ith word in document d
y_d	the class label of document d
$m_{z,w}$	number of words w which are assigned with topic z
$n_{d,z,z'}$	number of topics z followed by z' in document d
$\phi_{d,z}$	the multinomial distribution of topics w.r.t topic z in document d, those distributions constitute a topics-transition matrix Φ_d
ψ_z	the multinomial distribution of words w.r.t. topic z
α	Dirichlet prior of $\phi_{d,z}$
β	Dirichlet prior of ψ_z
θ_k	regression coefficients of class k

Figure 1: Notations used in Supervised Topic Transition model.

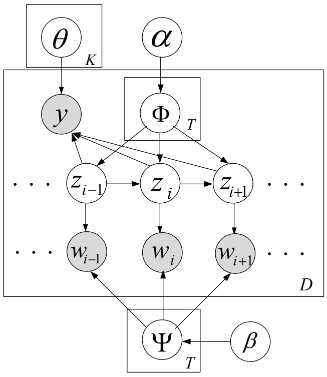


Figure 2: A graphical representation of the proposed Supervised Topic Transition Model. The gray nodes represent observed variables. The edges represent direct probabilistic interaction between the linked variables.

The graphical representation of STT is shown in Fig. 2. Consider step 3 of the generative process. We assume the class label for each document is drawn from a generalized logistic function with input given by the empirical distribution of topic transitions. This representation provides the flexibility of encoding arbitrary topic features while the output of the model always results in *well-calibrated* probabilities. This setting is inspired by sLDA, yet with an improvement. In sLDA, a response variable for each document is real valued and drawn from a linear regression. However, a continuous response is not appropriate for our goal of building a classifier. In STT, we exploit the generalized logistic function to supervise the topic model, which provides an important multi-class extension of the sLDA framework.

During the estimation process, class labels are observed. The labels are used to train a logistic regression model, which in turn provides an oracle and induces a subtle refinement of latent topics. As we shall see in Sect. 2.1, such reciprocal refinement is explicitly represented in the sampling formula and in the training algorithm. Another strong argument for coupling a logistic regression model is the discriminative power on unseen data. Given an unlabeled document d, the predicted label is $y_d = \arg\max_{y\in\{1,\cdots,K\}} P(y|\bar{z},\theta)$.

2.1 Parameters Estimation

Given a document collection, the aim is to estimate a topic transition matrix ϕ for each document, multinomial distributions of words for each topic ψ, as well as the regression coefficient θ shared by the collection. In many state-of-art topic models, Gibbs sampling is used to perform parameter estimation, whereas in logistic regression, gradient descent is the method of choice. Although the mechanisms behind these two algorithms are different, both algorithms are iterative methods. In this section, we present an iterative algorithm that combines Gibbs sampling (for estimating ϕ and ψ) and gradient descent (for estimating θ)

2.1.1 Gibbs sampling step

For every word in the collection, we sample the topic assignment from the following distribution:

$$P(z_{d,i}|\mathbf{w}, \mathbf{z}_{\neg(d,i)}, \mathbf{y}, \alpha, \beta, \theta) \propto$$

$$\overbrace{\exp(\sum_{z=1}^{T} \theta_{y_d,z,z_{d,i}} \bar{n}_{d,z,z_{d,i}}) \sum_{k=1}^{K} \exp(\sum_{z,z' \neq z_{d,i}}^{T} \theta_{k,z,z'} \bar{n}_{d,z,z'})}^{\text{logistic regression}} \quad \times$$

$$\underbrace{\left[n_{d,z_{d,i-1},z_{d,i}} + \alpha\right] \times \frac{m_{z_{d,i},w_{d,i}} + \beta}{\sum_{v=1}^{V} m_{z_{d,i},v} + V\beta}}_{\text{standard posterior}},$$

where counts m and n are counted except for $z_{d,i}$ (the Gibbs sampling derivation is provided in the Appendix). Notably, this Gibbs sampling formula consists of two parts: an exponential component from logistic regression where θ is fixed, and a standard posterior from the unsupervised topic model. After each iteration i, we can obtain the estimates of ψ and ϕ by calculating the followings:

$$\psi_{z,v}^{(i)} = \frac{m_{z,v} + \beta}{\sum_{v=1}^{V} m_{z,v} + V\beta}, \quad \phi_{d,z,z'}^{(i)} = \frac{n_{d,z,z'} + \alpha}{\sum_{z'=1}^{T} n_{d,z,z'} + T\alpha}. \quad (2)$$

The remaining task is estimating the regression coefficient θ.

2.1.2 Gradient descent step

Essentially, we want to use the topic assignments as features to train the logistic regression model after each iteration. The derivation of the gradient with respect to θ in the STT model is same as in the multi-class logistic regression model. We introduce y_{dk}, which follows a coding scheme as:

$$y_{dk} = \begin{cases} 1 & \text{if } y_d = k \text{ (document } d \text{ is labeled class } k\text{)}, \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

Consequently, the update function of θ is given by:

$$\theta_{k,z,z'}^{(i+1)} \leftarrow \theta_{k,z,z'}^{(i)} + \lambda \sum_{d=1}^{D} (y_{dk} - p_{dk}^{(i)}) \bar{n}_{d,z,z'}, \quad (4)$$

where λ is the learning rate, $p_{dk}^{(i)}$ is the predictive probability of document d labeled with class k in ith iteration, which has been given in (1). The new θ is used in the next Gibbs sampling step.

2.1.3 Summary

Putting all together, our training algorithm alternates between Gibbs sampling (2) and gradient descent (4). The two procedures affect each other by updating $\bar{n}$ and θ iteratively. The advantage of modeling sequences and labels jointly is twofold. First, the label subtly directs the topic evolution by minimizing the error value on classification. Second, the random sampling avoids the local optima that plagues gradient descent.

2.2 Parallelized STT

The computational complexity of Gibbs sampling in each round is determined by the number of topics multiplied by the total number of word occurrences in the training set, that is $\mathcal{O}(T \sum_{d=1}^{D} N_d)$. On a large-scale document collection, the standard Gibbs sampling is computationally infeasible. We therefore implement the training algorithm for the STT model in a parallel manner, which follows the idea of AD-LDA model [12]. The complete training algorithm is summarized in Algorithm 1:

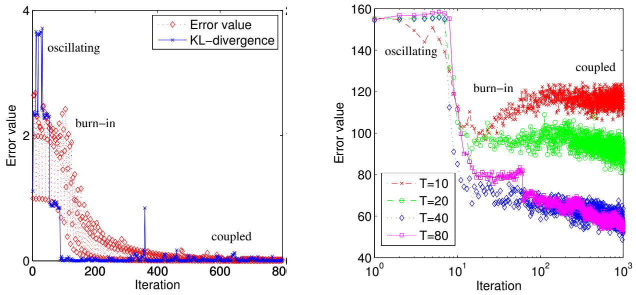
Algorithm 1 Parallelized training algorithm for STT model.

```
Input: Model parameters: $\alpha, \beta, \lambda, T$; Document collection: $D$; Available processors: $P$
Initialize $\theta$ randomly
Partition $D$ into $D^{|1}, \cdots, D^{|P}$
for all processor $p \in P$ do
    Initialize the topic assignments randomly
    Compute $m_{z,w}^{|p}$ and $n_{d,z,z'}^{|p}$
end for                                                      ┐ One iteration
repeat                                                       ┐ Gibbs sampling step
    for all processor $p \in P$ do
        for all document $d \in D^{|p}$ do
            for all word $w_{d,i}$ in document $d$ do
                Sample $z_{d,i}$ by (2)
            end for
        end for                                              ┘
    end for
    $m_{z,w} \leftarrow \sum_{p=1}^{P} m_{z,w}^{|p}$
    repeat                                                   ┐ Gradient descent step
        for all processor $p \in P$ do
            Compute $\nabla^{|p}$ by (4)
        end for
        $\nabla \leftarrow \sum_{p=1}^{P} \nabla^{|p}$, $\theta \leftarrow \theta + \lambda\nabla$
    until converged                                          ┘
until converged                                              ┘
return $\Psi, \Phi$ by (2) and $\theta$
```

2.3 Model convergence

As the logistic regression model is trained simultaneously with the Gibbs sampling procedure, it might not be obvious to reader that the training algorithm of the STT model will converge in general to a useful result. Therefore, we first employ a toy example to provide some insights into the convergence of STT. We define 3 topics over 8 system calls and create 5 programs artificially, three of the programs are labeled as harmless, the other two are malicious. We trained the STT model with this toy example and recorded the logistic regression error value $\sum_{d=1}^{D} (y_d - p_d)^2$ and the KL-divergence $\sum_{z=1}^{T} \Psi_z \log(\Psi_z / Q_z)$ after each iteration. The former gives an idea of the convergence of the logistic regression model. The latter measures the distance between the model's estimate Ψ_z and the true distribution Q_z.

Figure 3(a) shows these two measurements as a function of the iterations while training the model. One can ob-



(a) Error value and KL-divergence versus number of iterations. The model is trained on a toy example of five documents and three topics.

(b) Error value versus number of iterations. The models are trained on the real-world data set with 10, 20, 40 and 80 topics.

Figure 3: Convergence behavior of the STT model on a toy example (left) and a real-world data set (right).

serve, that the training process is roughly divided into three phases. In the first 100 iterations, the error value and the KL-divergence are “oscillating”. The model then comes to a “burn-in” phase, starting from 100 to 500 iteration, where both Gibbs sampling and gradient descent finds their way in parameter space. In this period, the KL-divergence is decreasing over time and the error value of the logistic regression is damped down. After 600 iterations, the model finally reaches an equilibrium — the “coupled” phase — where the KL-divergence and error value finally converged. In Fig. 3(b) the error value of the real-world data set with different number of topic is plotted. One can observe, that the model is $50-100$ iterations in the “oscillating” phase, followed by approximately 1000 iterations staying in the “burn-in” phase, and finally converges after approximately 1500 iterations. Additionally, one can observe in Fig. 3(b) that, the more topics the more accurate is the descriptive power of STT model, thus the smaller the error value of the logistic regression.

In standard Gibbs sampling, there is no obvious clue to tell the convergence of the algorithm, the sampling is often stopped after a desired number of iterations. In our model, a convergent Gradient descent implies a convergent Gibbs sampling. Consequently, by observing the error value, we can tell whether or not the training algorithm is convergent. This feature allows us to avoid unnecessary training steps and saves a lot of time when training on large-scale data set.

3. EXPERIMENTS

Due to the fact that most malicious programs exist on the Windows operating system (OS), we focus in this work on Windows programs. For collecting system calls of harmless and malicious programs a tool for Windows OS is developed which hooks in the OS and allows to gather system calls of executable programs. We collected system call sequences from 3048 programs[1] in 8 categories: Harmless, Email-worm, IM-worm, IRC-worm, Net-worm, Backdoor, Trojan and Others (i.e. Badjoke, HackTool etc.). In total, system call sequences of 168 harmless programs and 2880 malicious programs are collected. After pre-processing[2] the data, 34, 007, 743 system calls are in total gathered.

We present the experimental results from three perspectives. First, we interpret the uncovered patterns from system call sequences. Second, we use classification accuracy to numerically evaluate the STT model. Finally, we examine the performance of the parallelized training algorithm on large-scale data and analyze the bottleneck of it[3].

3.1 Analysis of Latent Topics

In this section the latent information learned in the STT model are investigated. In Table 1, eight topics are depicted which are found in a 40-topics run on the real-world data set (2, 000 Gibbs sampling iterations, symmetric priors $\alpha = 0.1$, $\beta = 0.01$ and $\lambda = 0.1$). The evolved topics are quite expressive[4]. Topic 1 provides a summary of the window interface handler, topic 12, 32 and 33 are related to graphics, process and memory handling and RPC (Remote procedure call), respectively. Besides, we also notice that some topics give extremely salient descriptions. For instances, in topic 14, the sum of the probabilities of `ReadFile` and `WriteFile` is 0.99; in topic 32 the single word `ReadProcessMemory` has a probability of 0.86. This phenomenon is due cyclically invoking system calls. It is clear that `ReadFile` and `WriteFile` are frequently invoked together, in most cases, cyclically in a loop. Similarly, one has to invoke `ReadProcessMemory` repeatedly to get a sufficient range of memory of a specified process. In general, the STT model is capable to capture co-occurrence patterns in sequences. However, in the special case when a word often occurs repetitively or two words frequently occur as repetitive pairs, then the STT model will give sharp and sparse topic results.

In Table 2, the topic transitions learned from nine programs are reported. We assign names for each topic by hand as we did in Table 1. For each program the top 6 topic transitions with highest probability are listed. By studying the learned topic-transitions, one can reveal the behavior of a program. Consider for example the program “mspaint.exe”. One can observe, that the frequent topic transition Graphics → Graphics implies a behavior such as “keep drawing pictures on device”. Another example is “Net.Worm.Lovesan”, which exploits the Windows RPC flaw to spread itself. The infecting and propagating behavior of “Net.Worm.Lovesan” is clearly reflected in its topic transitions.

Additionally, we visualized eight learned transition matrices (see Fig. 4) to study the functional similarity of programs. It is interesting to observe that malicious variations of the same stem (e.g. “IRC-Worm.Golember.p” and “IRC-Worm.Golember.u”) have similar topic-transition matrices. Since the STT model reveals the “behavior” in lower

[1] Available at `http://vx.netlux.org`.

[2] System call sequences of length smaller than 50 are omitted, as the length is not sufficient to characterize program behavior.

[3] The processed data set and a python-implementation of parallel STT are available at `http://www.sec.in.tum.de/~stibor/xiao/data+code.zip`.

[4] The description of each system call can be found on `http://msdn.microsoft.com/en-us/library/`.

Table 1: Eight topics from a 40-topic run of the STT model on the real-world data set. The 10 most probable words in each topic are depicted. Observe, that the STT model groups congeneric system calls together. For the sake of example the topic names are created by hand.

#1 Window Message	Prob.	#2 File Seek	Prob.	#5 Memory Control	Prob.	#12 Graphics	Prob.
PeekMessageA	.38	SetFilePointer	.58	VirtualAllocEx	.15	SelectPalette	.23
FindWindowA	.36	_llseek	.15	VirtualAlloc	.14	SelectObject	.18
GetCurrentProcessId	.11	GetFileSizeEx	.06	VirtualQueryEx	.12	GetVersion	.17
IsWindowVisible	.07	GetFileSize	.06	VirtualQuery	.11	SetDIBitsToDevice	.07
GetFileAttributesW	.04	CloseHandle	.04	GetCharABCWidthsW	.09	CreateCompatibleDC	.06
GetFileAttributesA	.04	MapViewOfFileEx	.02	VirtualFreeEx	.07	DeleteDC	.06
GetVersionExA	.00	MapViewOfFile	.02	VirtualFree	.07	SetDIBits	.06
TranslateMessage	.00	CreateFileMappingW	.02	FormatMessageA	.03	SetMapMode	.02
Sleep	.00	CreateFileMappingA	.01	lstrlenW	.03	FindAtomW	.01
SafeArrayGetDim	.00	UnmapViewOfFile	.01	GetProcAddress	.03	GlobalLock	.01
#14 File IO (Win32)	Prob.	#32 Process Memory	Prob.	#33 RPC Sync.	Prob.	#39 Registry Handler	Prob.
ReadFile	.63	ReadProcessMemory	.86	I_RpcRequestMutex	.14	GetProcAddress	.41
WriteFile	.36	VirtualQueryEx	.04	I_RpcClearMutex	.14	RegOpenKeyExA	.07
WriteConsoleA	.00	OpenProcess	.02	InterlockedIncrement	.10	RegQueryValueExA	.05
CloseHandle	.00	WriteProcessMemory	.02	InterlockedDecrement	.08	RegCloseKey	.04
_lclose	.00	CloseHandle	.02	GetCurrentThreadId	.06	RegEnumKeyExA	.03
RegOpenKeyA	.00	WideCharToMultiByte	.01	TlsGetValue	.04	LoadLibraryExW	.02
CreateFileW	.00	LocalFree	.01	InterlockedCmpExg	.04	LoadLibraryExA	.02
CreateFileA	.00	LocalAlloc	.01	RegOpenKeyExW	.02	LoadLibraryA	.01
wvsprintfW	.00	VirtualProtectEx	.00	CompareStringW	.02	RegEnumKeyA	.01
wvsprintfA	.00	VirtualAllocEx	.00	GetThreadLocale	.02	RegCreateKeyExA	.01

Table 2: Topic transitions learned from STT model. Four harmless programs: bootvrfy.exe (boot check), clipbrd.exe (clipboard control), dvdplay.exe (a DVD player) and mspaint.exe (a painting program). Five malwares: IM.Worm.Opanki, Email.Worm.NetSky, Email.Worm.Roron, Net.Worm.Lovesan and Net.Worm.Mytob. For each document, STT models has 40×40 topic transitions in total, we only report the top 6 transitions with highest probability.

bootvrfy.exe	clipbrd.exe	dvdplay.exe
Registry Read → Process Memory	String Handler → Locale Language	Process Status → GUI Sync.
Process Memory → Registry Handler	Process Memory → Unicode Handler	Locale Language → String Handler
File Delete → Registry Read	File Seek → RPC	Process Memory → Process Memory
Call DLL Func. → File Delete	Timer → File Copy	Graphics → Graphics
Registry Handler → Registry Handler	Call DLL Func. → Call DLL Func.	Thread Read → Error Handling
Registry Handler → Registry Read	Message Handler → GUI Sync.	Timer → Thread Sync.
mspaint.exe	IM.Worm.Opanki	Email.Worm.NetSky
Graphics → Graphics	String → GUI Sync.	Message Sync. → Process Memory
GUI Sync. → GUI Sync.	Thread Sync. → File Delete	RPC → Load Resources
Process Memory → Thread Sync.	Process Memory → Registry Read	String Handler → Locale Language
File IO Win32 → File Copy	File Delete → Process Memory	Graphics → Timer
File Seek → File IO Win32	Call DLL Func. → Registry Read	Error Handling → Registry Edit
Memory Control → Thread Sync.	GUI Sync. → Call DLL Func.	Memory Control → Registry Handler
Email.Worm.Roron	Net.Worm.Lovesan	Net.Worm.Mytob
File Copy → File Copy	Registry Edit → Registry Edit	File Delete → Thread Sync.
Process Status → File Delete	RPC → RPC	Registry Read → Registry Read
File Search → Registry Handler	File Search → File Search	File IO Win32 → Process Memory
GUI Sync. → Thread Sync.	GUI Sync. → Call DLL Func.	Memory Control → Registry Handler
String Handler → Thread Sync.	Registry Handler → Registry Handler	File Seek → File IO Win32
Message Handler → File Delete	Load Resources → Load Resources	Process Memory → Registry Read

dimensions by introducing latent topics, we are able to compare two programs without noisy influence in system call sequences. In this low "behavior" space, the original different variations are represented similarly. This representation helps classifying the harmless programs from malicious programs, and indeed this can be verified in the next section.

3.2 Classification

We study the classification performance of the STT model and compare it to a SVM which is fed with different input features. More specifically, we use the unigram and bigram model, where we first select the words with the highest frequency as features, thus each document is represented by a vector of relative frequencies. These frequency vectors are then used as input features for the SVM. We build two SVMs with 3560 unigram features[5] and 6400 bigram features, respectively. Moreover, we use $P(z|d)$ from the LDA model as a feature vector for the SVM. These three baselines are denoted as Uni+SVM, Bi+SVM and LDA+SVM, respectively. As STT model has discriminative power, it can be directly used to classify documents. In detail, we first sample a topic for each word in the test set by (2) as performed

[5] The number of unique system calls in this collection is 3560, thus the maximum size of unigram features is restricted to 3560.

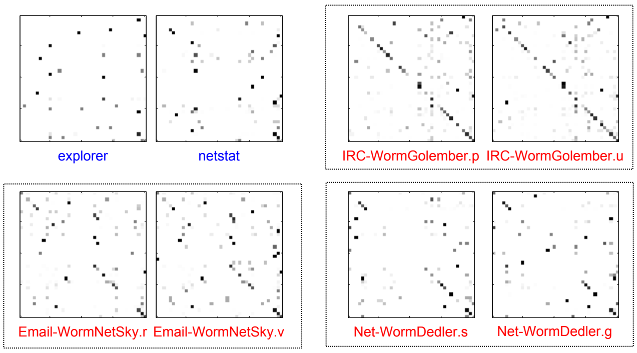


Figure 4: Visualized topic transition matrices of harmless and malicious (boldface) programs. Each matrix has 40-by-40 elements, where element (i, j) represents $P(z_j|z_i)$ with black being the highest probability and white being zero. Observe, that programs originating from the same malicious stem have similar topic transitions matrices.

in training algorithm, except that θ and $m_{z,w}$ are now fixed to the training result. After the sampling converged, we predict the label by $y_d = \arg\max_{y\in\{1,\cdots,K\}} P(y|\bar{z},\theta)$ for every document in test set. We also construct two classifiers based on the STT model. First, TT+LR[6] is built by separating the gradient descent from training algorithm. More precisely, we remove the supervision from STT, thus in each training iteration only the Gibbs sampling is performed. After the model converged, we feed the topic transition matrix to a logistic regression model. Another classifier is STT+SVM, where we feed the topic transition matrix learned from STT to train the SVM. We use LIBSVM [3] to build 1-vs-1 SVM classifier with RBF kernel and penalty term $C = 100$.For each model, a five-fold cross-validation is conducted and the following measurements are evaluated:

$$\text{Accuracy} = \frac{\#\textsc{malicious}\text{ classified as }\textsc{malicious}+\#\textsc{harmless}\text{ classified as }\textsc{harmless}}{\#\ \textsc{all}},$$

$$\text{False alarm rate} = \frac{\#\textsc{harmless}\text{ classified as }\textsc{malicious}}{\#\ \textsc{harmless}},$$

$$\text{Missing rate} = \frac{\#\textsc{malicious}\text{ classified as }\textsc{harmless}}{\#\ \textsc{malicious}}.$$

The results are illustrated in Fig. 5. Although the above measurements are prevalent in the community of malware detection, using such measurements we ignore the misclassification among different types of malware. To fully demonstrate the performance of different models, we also depict confusion matrices in Fig. 6.

One can observe, that the predictive power of STT is getting better when the number of topics is increasing, meanwhile both false alarm and missing rate are dropping. STT+SVM further refines the classification performance and yields a slightly higher accuracy and lower false alarm and missing rate. Moreover, by incorporating supervised information into topic model, STT finds a better latent space that can be used to predict topics and class labels, which yields better accuracy than TT+LR. This result further shows the effectiveness of our training algorithm. When the number of topics increase to 80, STT and STT+SVM enjoy impressive performance on three evaluations. Bi+SVM seems comparative with STT, yet it suffers from lower accuracy and higher false alarm rate. Uni+SVM keeps lowest missing rate which is surprising at the first glance, however, it is understandable with such a high false alarm rate. LDA+SVM gives a poor performance on multi-class task, which is only slightly better than Uni+SVM. Generally, the Markov family models, even a trivial bigram model, enjoy better accuracy in the task. This is due to such models can capture the sequential relationship between system calls, which are crucial in malware detection. On the other hand, the LDA model and unigram model are based on the bag-of-words assumption without encoding any sequential information. Apparently, only the pure co-occurrences information is not sufficient to express the behavior of a program, thus LDA and Unigram models result in poor accuracy. Moreover, we figured out, that LDA overfits at 40 topics. This also suggests that STT which combines aspects of both generative and discriminative classification, can handle more latent features than a purely generative model.

We also emphasize that precisely classifying malware is still a difficult problem. As depicted in Fig. 6, one can observe that almost all models failed to distinguish between Backdoor, Trojan and Others. Despite the internal similarity of these three kinds of programs, the accuracy might be increased by using more elaborate features rather than topic transition frequencies.

3.3 Parallel Performance

Having demonstrated STT's promising performance for the malware classification task, we parallelize it to gain speedup. We created four artificial data sets with $10,000$, $20,000$, $40,000$ and $80,000$ documents respectively, where each document consists of 1000 words. We trained STT on these data sets with 10 topics. The training is conducted on a Linux machine with 8 CPUs, each 2.7 Ghz and 64GB of memory in total. The average runtime of 50 iterations is recorded.

Fig. 7 shows the average time for one iteration as a function of the number of processors. By increasing the number of processors, we can significantly reduce the training time. For instance, for the collection of $80,000$ documents, the parallel STT model achieved approximately linear speedup of 7.4 on up to 8 processors. We also notice that the speedup is getting less effective, when the size of collection is getting small. This is observable for instance on $10,000$ documents, where the model yields a speedup of 4.6 on up to 8 processors. This phenomenon can be attributed to the IO overhead. Recall, that after each Gibbs sampling, a number of gradient descent steps are performed. After each gradient descent step, the worker nodes write the local gradient $\nabla^{|p}$ on the disk. The master node calculates the global gradient ∇ and writes the updated θ on this disk. The worker nodes then load the new θ and perform another step of gradient descent. When D is small, the computation of the local gradients does not

[6] Topic Transition + Logistic Regression. One can also view TT model as a Bayesian HMM, where a Dirichlet prior is added on each latent state.

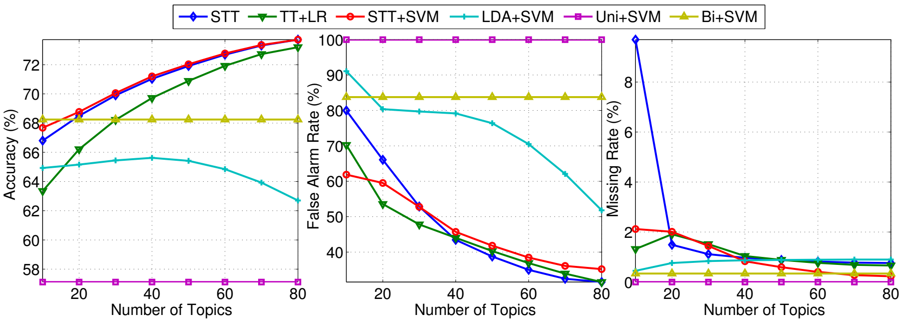
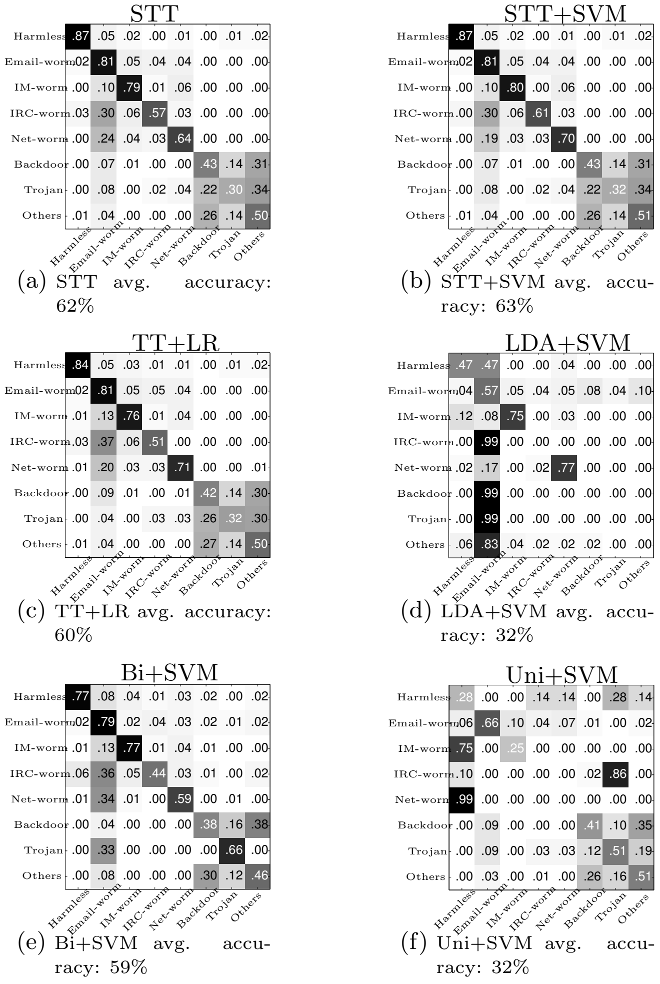


Figure 5: Classification result on real-world data set: (a) Accuracy (higher value is better), (b) False alarm rate (smaller value is better), (c) Missing rate (smaller value is better) of different models versus different number topics. Uni+SVM and Bi+SVM give constant performance as their feature size are fixed.



STT

	Harmless	Email-worm	IM-worm	IRC-worm	Net-worm	Backdoor	Trojan	Others
Harmless	.87	.05	.02	.00	.01	.00	.01	.02
Email-worm	.02	.81	.05	.04	.04	.00	.00	.00
IM-worm	.00	.10	.79	.01	.06	.00	.00	.00
IRC-worm	.03	.30	.06	.57	.03	.00	.00	.00
Net-worm	.00	.24	.04	.03	.64	.00	.00	.00
Backdoor	.00	.07	.01	.00	.00	.43	.14	.31
Trojan	.00	.08	.00	.02	.04	.22	.30	.34
Others	.01	.04	.00	.00	.00	.26	.14	.50

(a) STT avg. accuracy: 62%

STT+SVM

	Harmless	Email-worm	IM-worm	IRC-worm	Net-worm	Backdoor	Trojan	Others
Harmless	.87	.05	.02	.00	.01	.00	.01	.02
Email-worm	.02	.81	.05	.04	.04	.00	.00	.00
IM-worm	.00	.10	.80	.00	.06	.00	.00	.00
IRC-worm	.00	.30	.06	.61	.03	.00	.00	.00
Net-worm	.00	.19	.03	.03	.70	.00	.00	.00
Backdoor	.00	.07	.01	.00	.00	.43	.14	.31
Trojan	.00	.08	.00	.02	.04	.22	.32	.34
Others	.01	.04	.00	.00	.00	.26	.14	.51

(b) STT+SVM avg. accuracy: 63%

TT+LR

	Harmless	Email-worm	IM-worm	IRC-worm	Net-worm	Backdoor	Trojan	Others
Harmless	.84	.05	.03	.01	.01	.00	.01	.02
Email-worm	.02	.81	.05	.05	.04	.00	.00	.00
IM-worm	.01	.13	.76	.01	.04	.00	.00	.00
IRC-worm	.03	.37	.06	.51	.00	.00	.00	.00
Net-worm	.01	.20	.03	.03	.71	.00	.00	.01
Backdoor	.00	.09	.01	.00	.01	.42	.14	.30
Trojan	.00	.04	.00	.03	.03	.26	.32	.30
Others	.01	.04	.00	.00	.00	.27	.14	.50

(c) TT+LR avg. accuracy: 60%

LDA+SVM

	Harmless	Email-worm	IM-worm	IRC-worm	Net-worm	Backdoor	Trojan	Others
Harmless	.47	.47	.00	.00	.04	.00	.00	.00
Email-worm	.04	.57	.05	.04	.05	.08	.04	.10
IM-worm	.12	.08	.75	.00	.03	.00	.00	.00
IRC-worm	.00	.99	.00	.00	.00	.00	.00	.00
Net-worm	.02	.17	.00	.02	.77	.00	.00	.00
Backdoor	.00	.99	.00	.00	.00	.00	.00	.00
Trojan	.00	.99	.00	.00	.00	.00	.00	.00
Others	.06	.83	.04	.02	.02	.02	.00	.00

(d) LDA+SVM avg. accuracy: 32%

Bi+SVM

	Harmless	Email-worm	IM-worm	IRC-worm	Net-worm	Backdoor	Trojan	Others
Harmless	.77	.08	.04	.01	.03	.02	.00	.02
Email-worm	.02	.79	.02	.04	.03	.02	.01	.02
IM-worm	.01	.13	.77	.01	.04	.01	.00	.00
IRC-worm	.06	.36	.05	.44	.03	.01	.00	.02
Net-worm	.01	.34	.01	.00	.59	.00	.01	.00
Backdoor	.00	.04	.00	.00	.00	.38	.16	.38
Trojan	.00	.33	.00	.00	.00	.00	.66	.00
Others	.00	.08	.00	.00	.00	.30	.12	.46

(e) Bi+SVM avg. accuracy: 59%

Uni+SVM

	Harmless	Email-worm	IM-worm	IRC-worm	Net-worm	Backdoor	Trojan	Others
Harmless	.28	.00	.00	.14	.14	.00	.28	.14
Email-worm	.06	.66	.10	.04	.07	.01	.00	.02
IM-worm	.75	.00	.25	.00	.00	.00	.00	.00
IRC-worm	.10	.00	.00	.00	.00	.02	.86	.00
Net-worm	.99	.00	.00	.00	.00	.00	.00	.00
Backdoor	.00	.09	.00	.00	.00	.41	.10	.35
Trojan	.00	.09	.00	.03	.03	.12	.51	.19
Others	.00	.03	.00	.01	.00	.26	.16	.51

(f) Uni+SVM avg. accuracy: 32%

Figure 6: Comparisons using confusion matrices. Labels on the left are true labels (from top to bottom: Harmless, Email-worm, IM-worm, IRC-worm, Net-worm, Backdoor, Trojan and Others), labels at the bottom are predicted labels in the same order. Higher value represents better accuracy. The number of topics in STT, TT and LDA is fixed to 80. The average accuracy is computed by averaging of the diagonal values.

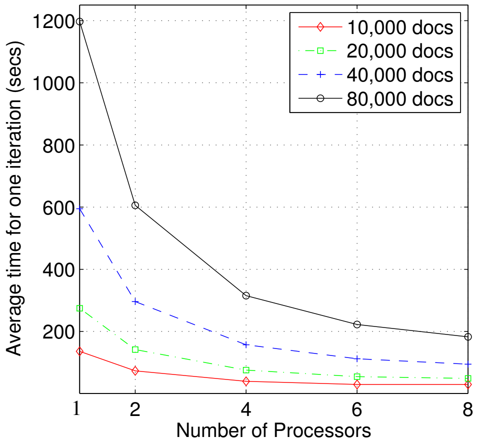


Figure 7: Scaleup result on artificial data for different document sizes.

take too much time, whereas frequent IO operations become a bottleneck of the training algorithm.

4. CONCLUSION

We have developed a new probabilistic topic model which is capable to recover the sequential patterns and perform malware classification simultaneously. This is achieved by jointly modeling sequences and class labels in the same latent space. Our training algorithm that alternates between Gibbs sampling and gradient descent is straightforward and easy to be extended. Comparing to previous approaches, we performed pattern discovery and malware classification in a single coherent model, rather than a stepwise pipeline. Experiments on a real-world data set suggested that the topics found by our approach are interpretable, and can be used to detect malicious programs. The comparative study showed that the our model outperforms other popular models on this classification task. Furthermore, we parallelized the training algorithm and demonstrated scalability with varying numbers of processors. In summary, our presented results are promising and underpin the effectiveness of probabilistic models for this kind of problem domain. Future work can address the scalability problem by means of online learning[7]. Furthermore by modeling higher-order dependencies of the system calls, deeper insights into the nature of malware can be obtained.

5. REFERENCES

[1] D. Blei and J. McAuliffe. Supervised topic models. *NIPS*, 20:121–128, 2008.

[2] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *JMLR*, 3:993–1022, 2003.

[3] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[4] M. Girolami and A. Kaban. Sequential activity profiling: latent Dirichlet allocation of Markov chains. *Data Mining and Knowledge Discovery*, 10(3):175–196, 2005.

[5] T. L. Griffiths, M. Steyvers, D. M. Blei, and J. B. Tenenbaum. Integrating topics and syntax. In *NIPS*, volume 17, pages 537–544. MIT Press, 2005.

[6] A. Gruber, M. Rosen-Zvi, and Y. Weiss. Hidden topic markov models. In *AISTATS*, 2007.

[7] M. D. Hoffman, D. M. Blei, and F. Bach. Online learning for latent dirichlet allocation. In *NIPS*, 2010.

[8] T. Hofmann. Probabilistic latent semantic analysis. In *Uncertainty in Artificial Intelligence (UAI)*, pages 289–296. Morgan Kaufmann, 1999.

[9] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6:151–180, 1998.

[10] S. Jha, C. Wang, D. Song, and D. Maughan, editors. *Malware Detection*. Advances in Information Security. Springer, 2007.

[11] J. Z. Kolter and M. A. Maloof. Learning to detect and classify malicious executables in the wild. *JMLR*, 7:2721–2744, 2006.

[12] D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed inference for latent dirichlet allocation. *NIPS*, 20:1081–1088, 2007.

[13] M. Steyvers and T. Griffiths. Probabilistic topic models. In T. K. Landauer, D. S. McNamara, S. Dennis, and W. Kintsch, editors, *Handbook of Latent Semantic Analysis*, chapter 21, pages 427–448. Lawrence Erlbaum Associates, 2007.

[14] H. M. Wallach. Topic modeling: beyond bag-of-words. In *ICML*, pages 977–984, 2006.

[15] X. Wang, A. McCallum, and X. Wei. Topical n-grams: Phrase and topic discovery, with an application to information retrieval. In *ICDM*, pages 697–702, 2007.

APPENDIX

Gibbs Sampling Derivation

We follow the notations declared in Table 1. The Gibbs Sampler draws a value to each latent variable by $z_{d,i} \sim P(z_{d,i}|\mathbf{w}, \mathbf{z}_{\neg(d,i)}, \mathbf{y}, \alpha, \beta, \theta)$, where $_{\neg(d,i)}$ indicates that the corresponding datum has been excluded. Our goal is to derive this distribution. We can rewrite the above probability using Bayes Rule as:

$$P(z_{d,i}|\mathbf{w}, \mathbf{z}_{\neg(d,i)}, \mathbf{y}, \alpha, \beta, \theta) = \frac{P(\mathbf{w}, \mathbf{z}, \mathbf{y}|\alpha, \beta, \theta)}{P(\mathbf{w}_{\neg(d,i)}, \mathbf{z}_{\neg(d,i)}, \mathbf{y}|\alpha, \beta, \theta)}. \tag{5}$$

The problem now reduces to derive the joint probability $P(\mathbf{w}, \mathbf{z}, \mathbf{y}|\alpha, \beta, \theta)$. The tricks of this manipulation are tripartite. Firstly, we introduce co-occurrence counters $m_{z,v}$ and $n_{d,z,z'}$ to replace the multinomial distributions. Secondly, we take advantage of conjugate prior to simplify the integrals. Moreover, the Euler integral[8] is used to make all remained integrals into product. We show that:

$$\begin{aligned}
P(\mathbf{w}, \mathbf{z}, \mathbf{y}|\alpha, \beta, \theta) &= \iint P(\mathbf{w}, \mathbf{z}, \mathbf{y}, \Psi, \Phi|\alpha, \beta, \theta)\, \mathrm{d}\Psi\, \mathrm{d}\Phi \\
&= \prod_{d=1}^{D} P(y_d|\mathbf{d}, \theta) \times \int \prod_{d=1}^{D} \prod_{i=1}^{N_d} P(w_{d,i}|\psi_{z_{d,i}}) \prod_{z=1}^{T} P(\psi_z|\beta) \mathrm{d}\Psi \\
&\quad \times \int \prod_{d=1}^{D} \left[\prod_{i=1}^{N_d} P(z_{d,i}|\phi_{z_{d,i-1}}) \prod_{z=1}^{T} P(\phi_{d,z}|\alpha) \right] \mathrm{d}\Phi \\
&= \prod_{d=1}^{D} P(y_d|\mathbf{d}, \theta) \times \int \prod_{z=1}^{T} \prod_{v=1}^{V} \psi_{z,v}^{m_{z,v}} \prod_{z=1}^{T} P(\psi_z|\beta) \mathrm{d}\Psi \\
&\quad \times \int \prod_{d=1}^{D} \prod_{z=1}^{T} \prod_{z'=1}^{T} \phi_{d,z,z'}^{n_{d,z,z'}} \prod_{d=1}^{D} \prod_{z=1}^{T} P(\phi_{d,z}|\alpha) \mathrm{d}\Phi \\
&= \prod_{d=1}^{D} P(y_d|\mathbf{d}, \theta) \times \int \prod_{z=1}^{T} \prod_{v=1}^{V} \psi_{z,v}^{m_{z,v}} \prod_{z=1}^{T} \left[\frac{\Gamma(\sum_{v=1}^{V} \beta_v)}{\prod_{v=1}^{V} \Gamma(\beta_v)} \prod_{v=1}^{V} \psi_{z,v}^{\beta_v - 1} \right] \mathrm{d}\Psi \\
&\quad \times \int \prod_{d=1}^{D} \prod_{z=1}^{T} \prod_{z'=1}^{T} \phi_{d,z,z'}^{n_{d,z,z'}} \prod_{d=1}^{D} \prod_{z=1}^{T} \left[\frac{\Gamma(\sum_{z=1}^{T} \alpha_z)}{\prod_{z=1}^{T} \Gamma(\alpha_z)} \prod_{z'=1}^{T} \phi_{d,z,z'}^{\alpha_{z'} - 1} \right] \mathrm{d}\Phi \\
&= \left[\frac{\Gamma(\sum_{v=1}^{V} \beta_v)}{\prod_{v=1}^{V} \Gamma(\beta_v)} \right]^{T} \times \left[\frac{\Gamma(\sum_{z=1}^{T} \alpha_z)}{\prod_{z=1}^{T} \Gamma(\alpha_z)} \right]^{DT} \times \prod_{d=1}^{D} P(y_d|\mathbf{d}, \theta) \\
&\quad \times \prod_{z=1}^{T} \int \prod_{v=1}^{V} \psi_{z,v}^{m_{z,v} + \beta_v - 1} \mathrm{d}\Psi \times \prod_{d=1}^{D} \prod_{z=1}^{T} \int \prod_{z'=1}^{T} \phi_{d,z,z'}^{n_{d,z,z'} + \alpha_{z'} - 1} \mathrm{d}\Phi \\
&\propto \prod_{d=1}^{D} [1 + \exp(-\sum_{z,z'}^{T} \theta_{z,z'} n_{d,z,z'})]^{-1} \prod_{z=1}^{T} \frac{\prod_{v=1}^{V} \Gamma(m_{z,v} + \beta_v)}{\Gamma(\sum_{v=1}^{V} m_{z,v} + \beta_v)} \\
&\quad \prod_{d=1}^{D} \prod_{z=1}^{T} \frac{\prod_{z'=1}^{T} \Gamma(n_{d,z,z'} + \alpha_{z'})}{\Gamma(\sum_{z'=1}^{T} n_{d,z,z'} + \alpha_{z'})}.
\end{aligned} \tag{6}$$

Finally, we have to show the full conditional probability of $z_{d,i}$ by substituting equation 6 in equation 5. Using chain rule and $\Gamma(x) = (x-1)\Gamma(x-1)$, an obvious reduction of fraction yields:

$$\begin{aligned}
P(z_{d,i}|\mathbf{w}, \mathbf{z}_{\neg(d,i)}, \mathbf{y}, \alpha, \beta, \theta) &\propto \frac{1 + \exp(-\sum_{z,z' \neq z_{d,i}}^{T} \theta_{z,z'} n_{d,z,z'})}{1 + \exp(-\sum_{z,z'}^{T} \theta_{z,z'} n_{d,z,z'})} \times \\
&\quad \frac{m_{z_{d,i}, w_{d,i}} + \beta_{w_{d,i}} - 1}{\sum_{v=1}^{V} (m_{z_{d,i}, v} + \beta_v) - 1} \times \frac{n_{d, z_{(d,i-1)}, z_{d,i}} + \alpha_{z_{d,i}} - 1}{\sum_{z=1}^{T} (n_{d, z_{(d,i-1)}, z} + \alpha_z) - 1}.
\end{aligned} \tag{7}$$

[7] Recently Hoffman et al. [7] proposed an online learning approach for latent Dirichlet allocation.

[8] $\int_{\sum_i x_i = 1} \prod_{i=1}^{N} x_i^{a_i - 1} \mathrm{d}^N \mathbf{x} = \prod_{i=1}^{N} \Gamma(a_i) / \Gamma(\sum_{i=1}^{N} a_i)$