INFOSHIELD: Generalizable Information-Theoretic Human-Trafficking Detection

Meng-Chieh Lee*
National Chiao Tung University
jeremy08300830@gmail.com

Catalina Vajiac*
Carnegie Mellon University
cvajiac@cs.cmu.edu

Aayushi Kulshrestha
McGill University & Mila
aayushi.kulshrestha@mail.mcgill.ca

Sacha Levy
McGill University & Mila
sacha.levy@mail.mcgill.ca

Namyong Park
Carnegie Mellon University
namyongp@cs.cmu.edu

Cara Jones
Marinus Analytics
cara@marinusanalytics.com

Reihaneh Rabbany
McGill University & Mila
rrabba@cs.mcgill.ca

Christos Faloutsos
Carnegie Mellon University
christos@cs.cmu.edu

***Abstract*—**

Given a million escort advertisements, how can we spot near-duplicates? Such micro-clusters of ads are usually signals of human trafficking. How can we summarize them, visually, to convince law enforcement to act? Can we build a general tool that works for different languages? Spotting micro-clusters of near-duplicate documents is useful in multiple, additional settings, including spam-bot detection in Twitter ads, plagiarism, and more.

We present INFOSHIELD, which makes the following contributions: (a) *Practical*, being scalable and effective on real data, (b) *Parameter-free and Principled*, requiring no user-defined parameters, (c) *Interpretable*, finding a document to be the cluster representative, highlighting all the common phrases, and automatically detecting "slots", i.e. phrases that differ in every document; and (d) *Generalizable*, beating or matching domain-specific methods in Twitter bot detection and human trafficking detection respectively, as well as being language-independent finding clusters in Spanish, Italian, and Japanese. Interpretability is particularly important for the anti human-trafficking domain, where law enforcement must visually inspect ads.

Our experiments on real data show that INFOSHIELD correctly identifies Twitter bots with an F1 score over 90% and detects human-trafficking ads with 84% precision. Moreover, it is scalable, requiring about *8 hours* for *4 million* documents on a stock laptop.

***Index Terms*—Anomaly Detection, Text Mining, Clustering Methods, Minimum Description Length (MDL), Anti-Human Trafficking**

I. INTRODUCTION

Given many documents, the majority of which do not belong to any cluster, how can we find small clusters of related documents? The driving application is human trafficking detection, where escort ads that are very similar are usually a sign of trafficking.

Finding related documents is a problem with numerous applications, such as search engines, plagiarism detection, mailing-address de-dupliclation, and more.

In this paper, we develop INFOSHIELD, a general, information-theory based method, and we illustrate its generality, effectiveness and scalability on two settings: escort advertisements, and Twitter data (both English as well as Spanish).

A. Application to the Human Trafficking Domain

While INFOSHIELD is general, our main motivation is near-duplicate detection and summarization in escort advertisements. Human trafficking (HT) is a dangerous societal problem which is difficult to tackle. It is estimated that there are 24.9 million people trapped in forced labor, 55% of which are women and girls accounting for 99% of victims in the commercial sex industry [1]. The majority of victims are advertised online [2] and 56% of victims have no input on ad content [2]. The average pimp has 4-6 victims [3]. Thus, the majority of ads suspected of HT are written by one person, who is controlling ads for 4-6 different victims at a time. By looking for small clusters of ads that contain similar phrasing, rather than analyzing standalone ads, we're finding the groups of ads that are most likely to be organized activity, which is a strong signal of HT.

Currently, law enforcement looks for HT cases manually, often one at a time. Our proposed INFOSHIELD will help them save time by detecting micro-clusters of similar ads, grouping them, and summarizing the common parts, as shown in Figure 1, which depicts Twitter data – we refrain from showing escort ad results for the victims' safety.

B. Application to Twitter Bot Detection

Detection of organized activity also has a clear application to bot detection; given millions of tweets, most of which come from legitimate users, how can we find tweets that exhibit bot-like behavior? The simplest kind of bot behavior is spamming, i.e. posting tweets that are almost or exactly identical in text, to increase visibility. Bot detection has been well-studied, but the majority of algorithms use manually crafted features that are specific to certain platforms, for example, the number of retweets. Our goal is to find near-duplicates in any application, which includes social media platforms containing text, such as Twitter. This particular application benefits from a vast amount of publicly available data.

* Both authors contributed equally to this work.

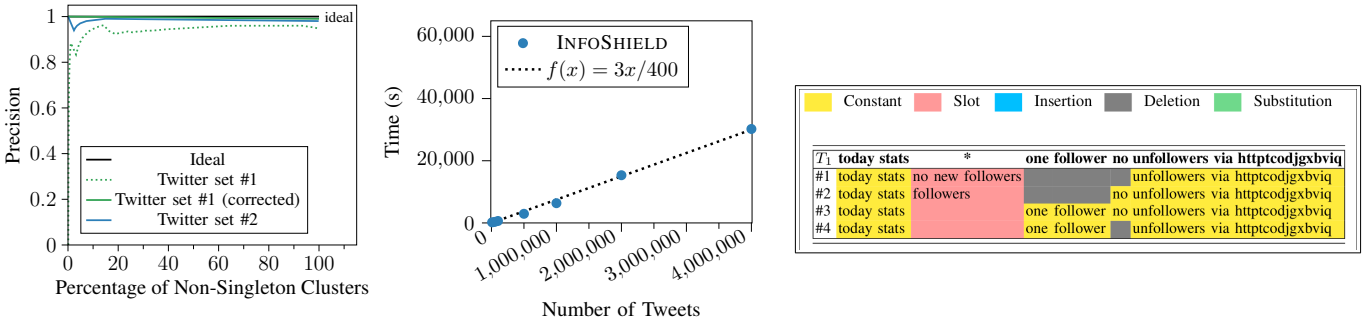


Fig. 1: INFOSHIELD *works:* being precise (left), scalable (middle), and interpretable (right), detecting and visualizing slots (in red), i.e. portions of tweets that highly differ between otherwise duplicate documents.

C. Our Method

Our first insight is to formalize the problem with information theory, and use the Minimum Description Length (MDL) principle to find good templates, which represent cluster text, with "slots", i.e. parts of the template that differ for each document. We mark slots with red highlights in Figure 1-right. We then use this summary to visualize the cluster. INFOSHIELD is *parameter-free*, since MDL can automatically pick the best choice of parameter values for any algorithm by choosing the combination with the shortest compression length. This is the INFOSHIELD-FINE part of our method.

The second insight is a novel pre-processing method, INFOSHIELD-COARSE, that dramatically improves scalability to be quasi-linear, by (a) eliminating single-copy documents/ads and (b) grouping the rest in coarse, but mainly homogeneous, clusters.

The resulting INFOSHIELD has a long list of desirable properties: It is

- *Practical*, processing *1 million* documents within *2 hours* on a Dell Alienware laptop with 32GiB RAM and i7 processor running Arch Linux; and correctly identifying Twitter spambots with an F1 score of 90% or higher,
- *Parameter-free & Principled*, requiring no user-defined parameters thanks to the Minimum Description Language principle,
- *Interpretable*, providing a clear visualization and summarization of the discovered micro-clusters.
- *Generalizable* and *domain independent* – we show results on two diverse areas, namely, Twitter data, and HTdata; as well as on multiple languages, i.e. Spanish, Italian, English.

Reproducibility: Our code is open-sourced at https://github.com/mengchillee/InfoShield. The HT dataset is available to researchers after NDA (email Dr. Cara Jones cara@marinusanalytics.com). The Twitter datasets are publicly available (see [4]).

II. BACKGROUND AND RELATED WORK

There is a lot of work on HT detection, document clustering, and multiple sequence alignment, and we group it in the following sub-sections.

A. Human Trafficking Detection

Some previous works try to classify whether or not a particular advertisement is suspected of HT [5], [6], [7], [8]. For instance, HTDN [7] proposes a supervised deep multimodal model trained on 10K manually labeled ads. Their results are later improved, on the same data, using an ordinal regression neural network [9]. Unfortunately, due to the adversarial nature of escort advertisements, these predefined or learned features don't stay relevant over time. These labeled ads are also expensive to obtain (requiring the precious time of domain experts) and are error-prone, as will be discussed in Section V. Moreover, inspecting ads individually, we might overlook ads that are part of an organized activity but do not stand out on their own. Therefore, unsupervised algorithms that find *groups* of organized activity are preferred in this domain. Template Matching [10] exploits the above insight, being the first anti HT method to our knowledge to perform clustering. However, the interpretability of clusters is limited, and the algorithm isn't scalable.

B. Social Media Bot Detection

Most efforts in detecting bots in social media platforms are formulated as supervised classification based on features from users and the content they post [11], [12]. Fewer works look for anomalies or fraud in networks, rather than in text, for instance [13]. A notable method, Botometer [14], formerly called BotOrNot, is an online service that provides a score of likelihood that a particular user is a bot. Since it is the only state-of-the-art method with public access to the implementation, we will use it as a baseline for our experiments in Section V. [4] gives a more comprehensive overview of Twitter bot detection methods, and also provides the dataset we will use in Section V-A1. Very few works focus on detecting *organized activity* - groups working together to mislead people about who they are and what they are doing, which is a rising issue [15]. ND-Sync [16] finds a related but different type of behavior, i.e. "retweet spam", where groups of multiple users exhibit organized behavior by consistently upvoting a particular user's tweets.

C. Document Embedding and Clustering

Much work has been done to represent documents in a machine-understandable format. The most widely-used approaches to represent documents include bag of words [17] and term frequency-inverse document frequency (tf-idf) [18]. These methods are commonly used for plagiarism detection [19], [20], [21], [22], which is a similar setting to near-duplicate detection. However, none of these methods do visualization or ranking, and some assumptions do not work in our case, i.e. [22] assumes documents consist of multiple lines, which is not the case for tweets or the majority of escort advertisements.

Unsupervised word vector models such as Word2Vec [23], Doc2Vec [24], and FastText [25] assume that words occurring in the same context tend to have similar meaning, with much success. However, these methods require large amounts of time and data to train. Even when trained using large datasets from Twitter data and the HT domain, we find that these generic embedding methods do not perform as well, as shown in Section V.

Given any document embedding, we can choose from many clustering algorithms. Density-based clustering techniques are most relevant to finding small dense text clusters, such as DBSCAN [26], HDBSCAN [27], OPTICS [28], or k-means [29]. These are all powerful methods, but most of them are searching for large clusters, as opposed to micro-clusters that we care about,and none of them do slot-detection.

In Table I, we give several question-marks for clustering methods because some of the methods are scalable (k-means), while others are almost quadratic; some methods are parameter-free (G-means), but most are not.

Finding pairs of nearby points (or intersecting rectangles) is an old problem, under the name of "spatial joins" [30], [31]. However, these methods are best for low-dimensional spaces, since they use the R-tree [32] spatial access method.

D. Multiple-Sequence Alignment

Multiple-Sequence Alignment (MSA) is a well-studied area with an application to biology, for comparing DNA sequences. The Barton-Sternberg algorithm [33] is an early profile-based approach which aligns sequences by updating a profile sequence iteratively. However, profile-based approaches generate ambiguity among sequences. To solve this, [34] uses partial order graphs instead of profile sequences, which enables a base in dynamic programming to have multiple predecessors and successors.

Nature Language Processing (NLP) is another area benefiting from MSA. [35] applies MSA to learn the patterns of given word sequences by word lattices and rewrite the sentences. [36] focuses on aligning sentences by syntactic features to create the description for a particular fact. However, most of these methods highly rely on parameter tuning and English syntactical rules, assuming that all sentences are grammatically correct. This assumption does not hold for data on any social network or for escort advertisements, where misspellings and grammatical errors are common. Thus, these methods are not generalizable.

E. Minimum Description Length

The Minimum Description Length principle (MDL) [37] assumes that the best model $M \in \mathbb{M}$ for data D minimizes $C(M) + C(D|M)$, where $C(x)$ is defined as the cost, i.e. number of bits, needed to describe x losslessly. The main insight is that it penalizes both the model cost $C(M)$, as well as the encoding of errors/deviations from the model $C(D|M)$ - while several other methods ignore the model complexity.

MDL has been extremely successful in several data mining tasks [38], including decision trees (SLIQ [39]), graph mining (CrossAssociations [40]), time series segmentation and mining (AutoPlait [41]), string similarity [42], and many more applications. It formalizes the very intuitive "Occam's razor" idea: the simplest explanation for a phenomenon or dataset is the best explanation.

While all of the above methods have provided unique and interesting contributions, none have all of the same features as INFOSHIELD. Table I contrasts INFOSHIELD against the state of the art competitors.

Property \ Method	Clustering [26], [27], [28], [29]	Barzilay and Lee [35]	Shen et al. [36]	HDTN [7]	Template Matching [10]	INFOSHIELD
Practical- Scalable	?			✓	✓	✓
Practical- Effective	?	?	?	?	✓	✓
Practical- Ranked output	?			?	✓	✓
Parameter-free	?					✓
Principled						✓
Interpretable	?	✓	✓		✓	✓
Slot Detection		✓				✓
Generalizable	✓					✓

TABLE I: INFOSHIELD *matches all specs*, while competitors miss one or more of the features. "?" means that it depends on the specific clustering method, or that it is unclear from the original paper.

III. PROPOSED METHOD - THEORY

In this section we present the theory behind our proposed method.

A. Intuition and Theory

Our problem is split in the following parts: Given N documents, where we suspect that there are small clusters of organized activity:

1) Theory: how do we measure the goodness of a set of clusters, and

2) Algorithms: how do we quickly find clusters that describe patterns in the data concisely (INFOSHIELD-COARSE– Section IV-A) and then how do we refine and visualize these clusters (INFOSHIELD-FINE– Section IV-B).

Our MDL-based approach is best explained with examples.

Example 1 (Simple toy example)**.** *Suppose we have the documents of Table II in a particular cluster.*

How could we summarize them in a human-explainable form?

Doc#	Text
#1	This is a great soap, and the 5 dollar price is great
#2	This is a great chair, and the 10 dollar price is great
#3	This is a great hat, and the 3 dollar price is great

TABLE II: Simple toy example

One part of our proposed INFOSHIELD is to use *templates*, which consist of constant strings and variable strings, called slots. We depict slots with '*', following the Unix convention. We also allow the usual string-editing operations (insertions, deletions, and substitutions). Thus, for the above 3-ad example, a human (and our INFOSHIELD) would produce the template: "**This is a great *, and the * dollar price is great**" as shown in Table IV.

Let's also consider a more complicated cluster with multiple templates:

Example 2 (Full toy example)**.** *In addition to the documents of Example 1, suppose that we also have the documents of Table III:*

Doc#	Text
#4	This is great blue pen, and the 3 dollar price is so good
#5	I made 30K working on this job - call 123-456.7890 or visit scam.com
#6	I made 30K working from home - call 123-456.7890 or visit fraud.com
#7	Happy birthday to my dear friend Mike

TABLE III: Full, toy example

Doc # 4 belongs in T_1, but with one deletion (omitting "a"), one insertion (adding "so") and one substitution (replacing "great" by "good"). However, Docs #5-7 clearly do not belong to the same template. We now would expect to see two templates T_1 and T_2, with T_1 representing Doc #1-4, T_2 representing Doc #5-6, and Doc #7 does not belong to any template.

Furthermore, we would like to visualize the templates that we do find as follows:

In more detail, but still informal, INFOSHIELD should achieve lossless compression, with the cost being as follows:

1) *Model complexity* $C(M)$: the cost to encode the t templates we discover. In our working example, this would be the coding cost (roughly, the number of characters, below), for
   T_1: "This is a great *, and the * dollar price is great"
   T_2: "I made 30K working * - call * or visit *"

Constant Slot Insertion Deletion Substitution

T_1	**This is a**	**great**	*	**and the**	*	**dollar price is**	**great**
#1	This is a	great	soap,	and the	5	dollar price is	great
#2	This is a	great	chair,	and the	10	dollar price is	great
#3	This is a	great	hat,	and the	3	dollar price is	great
#4	This is	great	blue pen,	and the	3	dollar price is	so good

T_2	**I made 30k working**	*	**- call**	*	**or visit**	*
#5	I made 30k working	on this job	- call	123-456.7890	or visit	scam.com
#6	I made 30k working	from home	- call	123-456.7890	or visit	fraud.com

TABLE IV: Templates for Full, toy example.

2) *Data compression* $C(D|M)$: the cost to encode slot-values, insertions, and deletions, for each of the documents, with respect to its best template (or just the listing of the words in the document, if no template matches). Thus, for each document, we must store (a) the tokens in slots, (b) position and token for insertions, (c) position for deletions, (d) position and token for substitutions, and (e) the template-id that best matches the document. Table V shows the information we include in $C(D|M)$ for our running example.

Doc	Temp.	Slots	Ins.	Del.	Sub.
#1	T_1	{"soap", "5"}			
#2	T_1	{"chair", "10"}			
#3	T_1	{"hat", "3"}			
#4	T_1	{"blue pen", "3"}	12: "so"	3	13: "good"
#5	T_2	{"on this job", "scam.com"}			
#6	T_2	{"from home", "fraud.com"}			
#7	*N/A*	"Happy birthday to my dear friend Mike"			

TABLE V: Example encoding for $C(D|M)$

Notice that Docs #1-4 are compressed with much fewer characters when we use template T_1, since they have so many phrases in common.

The coding cost is roughly proportional to the number of characters we need to describe (1) and (2) above. More formally,

Definition 1. *[Total encoding cost] The total coding cost for a set of n documents with t templates is given by*

$$C = C(M) + C(D|M) \tag{1}$$

In Section III-B, we explain the exact cost for N documents and t templates more precisely. Then, in section IV, we propose algorithms on how to *discover* such a good set of templates.

We want to highlight that the separation of the cost function in Equation 1 from the algorithms makes INFOSHIELD extensible: we can use any and every optimization algorithm we want. The ones we propose in Section IV are carefully thought-out, and give meaningful results, but any other set of algorithms is fine to include – we can pick the solution with the best coding cost.

Furthermore, INFOSHIELD is parameter-free: any optimization algorithm minimizing total cost does not need user-defined parameters — we can try as many parameter values as we want, and pick the solution with the lowest cost.

B. Data Compression and Summarization

In this subsection, we give the details of the encoding cost in INFOSHIELD. Table VI provides symbols and definitions relevant to the encoding.

Symbols	Definitions
N	Total number of documents in D
t	Total number of templates
V	Number of words in vocabulary
T_i	i-th template
l_i	Length of template T_i
s_i	Number of slots in T_i
$\hat{l}_d$	Alignment length of data d
$w_{d,j}$	Number of words in j-th slot in aligned data d
e_d	Number of unmatched words in aligned data d
u_d	Number of substituted/inserted words in aligned data d
$\langle n \rangle$	$\approx 2 \lg n + 1$: universal code length for a non-negative integer
$\lg(L)$	$= \log_2(L)$: code length for integer i $(1 \leq i \leq L)$

TABLE VI: Symbols and definitions for INFOSHIELD-FINE

1) Template Encoding: We use the notation $\langle n \rangle$ for the coding cost of integer n, using the universal code length [43], that is $\langle n \rangle = \log^* n \approx 2 \times \lg n + 1$.

We also assume that we have V vocabulary words total and that each is encoded as an index, requiring $\lceil \lg V \rceil$ bits. For a length-l document, we need $\langle l \rangle$ bits to encode the number of words and $\lg V$ for each word, resulting the total cost $\langle l \rangle + l * \lg V$.

Definition 2 (Model encoding cost)**.** *The coding cost for t templates is given by*

$$C(M) = \langle t \rangle + \sum_{i=1}^{t} \langle l_i \rangle + l_i \lg V + (1 + s_i) \lg l_i \qquad (2)$$

Let's describe every term of the above definition:

- $\langle t \rangle$ - universal coding, for the number of templates T
- For each template T_i, we need:
  - $\langle l_i \rangle$ to encode the number of words in the i-th template
  - $\lg V$ for each word in T_i
  - $\lg l_i$ for the number of slots s_i in the template, and
  - $\lg l_i$ for the location of each slot.

Arithmetic Example 1. *The encoding cost for a single template T_i with 10 tokens and 2 slots is:*

$$\langle 10 \rangle + 10 \lg V + 3 \lg 10$$

2) Alignment Encoding: Given a template and a document that it describes, what is the best way to encode the document? The intuition is to encode insertions, deletions, and substitutions in the template, and the tokens in slots. For the templates, we need only encode the word-location of a mismatch, its type, and, for insertion/substitution, we encode the relevant word.

Definition 3 (Data encoding cost)**.** *The coding cost for N documents encoded with t templates is given by*

$$\begin{aligned} C(D|M) = N &+ l_d \times \lg V \\ &+ \sum_{i=1}^{t} \sum_{d \in D_i} (\lg t + \left\langle \hat{l}_d \right\rangle + \hat{l}_d \\ &+ e_d \lg \hat{l}_d + u_d \lg V + \sum_{j=1}^{s_i} \mathcal{S}(w_{d,j})), \end{aligned} \qquad (3)$$

where D_i denotes the data encoded by template T_i. We describe this definition in more detail:
Let D_U denotes the documents that do not match any template. The encoding cost for data $d \in D_U$ which is not encoded by template is simply computed by $l_d \times \lg V$. For the rest, the reasoning is as follows: Given a template T_i and a document $d \in D_i$, the alignment coding cost is:

- 1 bit for template flag yes/no
- $\lg t$ for template-id (if the flag is 'yes'):
- $\left\langle \hat{l}_d \right\rangle$ for length of the alignment
- 1 bit for each word in alignment if matched/unmatched
- $\lg \hat{l}_d$ for the location of each unmatched word
- $\lceil \lg 3 \rceil = 2$ bits for operation type of each unmatched word (insertion/deletion/substitution)
- $\lg V$ for word index in vocabulary if insertion/substitution
- $\mathcal{S}(w_{d,j})$ for the number of words $w_{d,j}$ in j-th slot:

$$\mathcal{S}(w_{d,j}) = 1 + \begin{cases} \langle w_{d,j} \rangle + w_{d,j} \lg V & \text{, if } w_{d,j} > 0 \\ 0 & \text{, otherwise} \end{cases} \qquad (4)$$

- repeat, for all other editing operations

Arithmetic Example 2. *The alignment encoding cost of Doc #4 by template T_1 (see Table IV), is the following:*

$$\begin{aligned} &\lg 2 + \langle 14 \rangle + 14 \\ &\quad + 3 \lg 14 + 2 \lg V + 2 \times (1 + \langle 1 \rangle + 1 \lg V) \end{aligned} \qquad (5)$$

3) Overall Encoding: Notice that we ignored the cost of encoding the vocabulary, since it would be the same for all sets of templates, and roughly the number of bytes to spell out all the vocabulary words, separated by a word-delimiter, such as a newline character. More accurately, this would be: $\langle V \rangle + V \times (l + 1) \times 8$ where l is the average word length, 8 bits per character, and 1 bit for the delimiter between words.

IV. PROPOSED METHOD - ALGORITHMS

How can we find templates that minimize our cost function in a scalable way? While the intuition described in Section III is correct, finding such templates is an expensive operation, being quadratic in the worst case. Thus, we first create reasonable clusters of related documents in a scalable way, using INFOSHIELD-COARSE, then work to find templates within each cluster using INFOSHIELD-FINE. If the average cluster size remains small, in comparison to N, then we process N documents in sub-quadratic time.

A. INFOSHIELD-COARSE

How do we quickly create coarse-grained clusters of documents with high text similarity? We start with document embedding, then perform clustering.

1) Document Embeddings: How do we generate a meaningful document embedding? We wish to capture similarity between documents that contain similar phrasing, but may have small variations (insertions, deletions, misspellings, etc). To this end, we first calculate the tf-idf weights for each phrase (n-gram)-document pair in the corpus. When calculating tf-idf, we consider phrases up to 5-grams. [1]

Then, for each document, we extract the top phrases with the highest tf-idf scores. By using tf-idf and limiting the number of phrases used, we only keep the most important phrases in the document that are unique to only a few advertisements, while ignoring commonly-used phrases. By making the number of phrases selected a function of input size, we reduce the risk of our results being heavily impacted by document length. Since some documents have a maximum length (i.e. tweets) but many do not, this helps to prevent INFOSHIELD-COARSE from being domain-specific.

2) Clustering: Now, how do we quickly create meaningful candidate clusters? We construct a bi-partite graph of documents and phrases. For any document i and phrase j, we construct an edge i, j if j is a top phrase in i. Once all documents are processed, we consider all connected components in G to be our coarse-grained clusters.

In the case that these clusters end up too large; an "unimportant" phrases combined documents that ideally should not be combined, we rely on INFOSHIELD-FINE to refine these clusters and split them if necessary. This is why INFOSHIELD-COARSE is very permissive, only requiring ads to share one important phrase to be connected.

Algorithm 1 shows more formally how to construct a document graph using INFOSHIELD-COARSE.

Data: N documents
Result: candidate clusters generated from N
initialize empty document-phrase graph
$G = (V_1, V_2, E)$;
forall *documents* d **do**
  forall *phrase* p *in FindTfidfPhrase(*d*)* **do**
    $E \leftarrow E \cup (d, p)$;
clusters $\longleftarrow$ FindConnectedComponents(G) ;

Algorithm 1: INFOSHIELD-COARSE

B. INFOSHIELD-FINE

Once we have coarse-grained clusters, how do we find templates and visualize the resulting clusters? Given data D containing multiple documents, split into coarse-grained clusters, the goal is to automatically find a template set M containing zero or more templates. Each template is expected to encode at least two documents. Within each coarse-grained cluster, the first task is to generate non-singleton candidate sets of documents and find potential templates. Next, we search for the best consensus document, i.e. the document that most represents the cluster, and detect possible slots by optimizing our cost function in Equation 3. We continue finding templates until we've processed all documents in a coarse-grained cluster, then move to the next cluster. We divide our algorithm into three major steps as follows:

1) **Candidate Alignment:** Identify the candidate set for a template and align all the documents in the set, using multiple sequence alignment (MSA).
2) **Consensus Search:** Search for the best consensus document in the alignment.
3) **Slot Detection:** Detect slots in the consensus document to generate a template.

To compute the MSA, we carefully choose to use Partial Order Alignment (POA) [34] as our alignment method for its effectiveness and efficiency. It is worth noting that INFOSHIELD-FINE can co-work with any off-the-shelf MSA approaches.

1) Candidate Alignment: Given data D from one cluster generated by INFOSHIELD-COARSE, containing multiple documents at iteration i, the candidate set for the template needs to be identified first. We first align all the documents $d \in D$ with the first document d_1 individually and then compute the cost $C(d|d_1)$ and $C(d)$ for every $d \in D$; if $C(d|d_1)$ is smaller than $C(d)$, meaning that d and d_1 have high similarity and can possibly be encoded by the same template, we add d into the set D_i containing all similar documents found in iteration i. Finally, we generate the alignment A_i by the POA method with all documents in D_i.

2) Consensus Search: After generating alignment A_i, how do we decide which tokens are part of the template, and which are insertions/deletions/substitutions? Keeping too many words in the template causes more unmatched operations (insertion/deletion/substitution); while keeping too few words hurts interpretability.

To solve this problem automatically, we turn it into an optimization problem by MDL. Function $Sel(A_i, h)$ is used to select the sub-alignment from the POA graph, where we only keep edges between words that occur more than h times in A_i. We aim to search for the best threshold h_i^* to generate the consensus of alignment with the lowest cost. The optimization problem can then be formed as follows:

$$h_i^* = \min_h C(D_i|Sel(A_i, h)) \tag{6}$$

Although our cost function is not convex, the optimization problem is only 1-dimensional, being relatively easy to solve. Hence we employ the Dichotomous Search algorithm [44] as our optimization method, where it returns the optimal solutions in most cases. The optimization algorithm is shown in Algorithm 2, where we iteratively shrink the search space to half. The consensus document $T_i^{'}$ only contains one sequence and no slot.

3) Slot Detection: Once we have a template, how do we find slots? Slots contain parts of documents which we expect

[1] Phrase length has little impact on results past $n = 5$: see Section V-E.

Data: An alignment A_i and a candidate set D_i
Result: A consensus document $T_i^{'}$
Initialize $h_L = 0, h_R = |D_i| - 1$;
while $h_L < h_R$ **do**
  $h_M \leftarrow (h_L + h_R)/2$;
  if
   $C(D_i|Sel(A_i, h_M - 1)) \leq C(D_i|Sel(A_i, h_M + 1))$
   then
    $h_R \leftarrow h_M - 1$;
  else
    $h_L \leftarrow h_M + 1$;
$T_i^{'} \leftarrow Sel(A_i, h_M)$;
Return $T_i^{'}$;

Algorithm 2: *Consensus-Search*

to differ, either in length or content, in the same location of each document. Slots inherently differ from unmatched words; instead of storing the location of each unmatched word per document as we would for unmatched words, we only store the location once, as part of the template.

Algorithm 3 shows how we do slot detection. We first recognize the operation types of words by each alignment $a \in A_i$, which are either insertions or substitutions. We identify which words each potential slot p contains in the given consensus document $T_i^{'}$. With this information, the computation of total cost with or without the slot p can easily be done. We only keep slots that decrease the total cost and store them in T_i.

4) Relative Length: To study the quality of compression by INFOSHIELD-FINE, we use relative length:

$$\text{Relative Length} = \frac{\text{Cost after compression}}{\text{Cost before compression}} \tag{7}$$

When relative length is close to 1, it means that the quality of compression is low; when it is close to lower bound, it means that the quality of compression is high, and the compressed documents are near-duplicate. For that reason, we derive the lower bound encoding cost of a cluster to study whether it is close to near-duplicate or not.

Lemma 1. *The lower bound encoding cost of a cluster by* INFOSHIELD-FINE *is*

$$\frac{t}{n} + \frac{1}{\lg V} \tag{8}$$

where t denotes the number of templates in the cluster, n denotes the number of documents in the cluster, and V denotes the number of words in vocabulary.

Proof. The encoding cost of n documents without template is $nl \lg V$. By Equation 2, we know that the encoding cost of t templates is $\langle t \rangle + t(\langle l \rangle + l \lg V + \lg l)$; and by Equation 3, we know that the encoding cost for each document with no unmatched words is $(1 + \langle l \rangle + l)$. We can then derive:

$$\frac{\langle t \rangle + t(\langle l \rangle + l \lg V + \lg l) + n(1 + \langle l \rangle + l)}{nl \lg V} \approx \frac{t \lg V + nl}{n \lg V} \approx \frac{t}{n} + \frac{1}{\lg V} \tag{9}$$

where l is a small constant value that is negligible. So the total encoding cost for n near-duplicate documents by t templates is approximately $\frac{t}{n} + \frac{1}{\lg V}$. ■

Data: A consensus document $T_i^{'}$, an alignment A_i, and a candidate set D_i
Result: A template graph T_i with slot(s)
Initialize P as a dictionary, $T_i = T_i^{'}$;
for $a \in A_i$ **do**
  $x = 0$;
  for $j = 1, .., l_a$ **do**
    if a_j *is an insert or substitution word* **then**
      $P[x] \leftarrow P[x] + 1$;
    else
      /* a_j is a matched or deleted word */
      $x \leftarrow x + 1$;
for $p \in P$ **do**
  if $C(D_i|T_i^{'}(p.slot \leftarrow True)) < C(D_i|T_i^{'})$ **then**
    $T_i \leftarrow T_i(p.slot \leftarrow True)$;
Return T_i;

Algorithm 3: *Slot-Detection*

5) Overall Algorithm: The overall algorithm of INFOSHIELD-FINE is shown in Algorithm 4. Given data D containing multiple documents from one cluster by INFOSHIELD-COARSE, we first initialize the template set $\mathcal{T}$ and the number of detected template i. At iteration i, we initialize alignment by the first document $d_0 \in D$. We compare with all other documents $d \in D$ to identify whether they should be encoded by the same template. After generating the alignment A_i and the data D_i that it encodes, we search for the best consensus sequence $T_i^{'}$ by optimizing the cost function. Then we detect the slots on the consensus sequence $T_i^{'}$ to generate template T_i. We include the T_i into our template set $\mathcal{T}$, and compute the total cost for both templates and data encoded by templates. If the total cost decreases by including T_i, we include it into $\mathcal{T}$ and update the total cost; otherwise, we treat D_i as noise. We run INFOSHIELD-FINE on every cluster generated by INFOSHIELD-COARSE, thus our final model M is $\mathcal{T}_1 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_m$, where m is the number of coarse clusters. It is worth noting again that INFOSHIELD-FINE is parameter-free, needing no human-defined parameters and optimizing for each template automatically.

Data: Data D consisting of multiple documents
Result: A template set $\mathcal{T}$
Initialize $\mathcal{T}, c^* = C(D), i = 1$;
while $|D| > 0$ **do**
  Initialize $A_i = d_1$ by the first document in D;
  Initialize candidate set D_i;
  for $d \in D[2:]$ **do**
    if $C(d|d_1) < C(d)$ **then**
      $D_i \leftarrow D_i \cup \{d\}$;
      $A_i \leftarrow MSA(A_i, d)$;
  $T_i^{'} \leftarrow ConsensusSearch(A_i, D_i)$;
  $T_i \leftarrow SlotDetection(T_i^{'}, A_i, D_i)$;
  $c \leftarrow C(\mathcal{T} \cup \{T_i\}) + C(D|\mathcal{T} \cup \{T_i\})$;
  if $c < c^*$ **then**
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{T_i\}$;
    $c^* \leftarrow c, i \leftarrow i + 1$;
  else
    Treat D_i as noise(s);
  $D \leftarrow D \backslash D_i$
Return $\mathcal{T}$;

Algorithm 4: INFOSHIELD-FINE

C. Complexity Analysis

Lemma 2. INFOSHIELD *is quasi-linear on the input size, taking time*

$$O(Ncl) + O(k_{max} N log(N) l^2) \tag{10}$$

where N is the number of documents, l is the (maximum) length of a document, m is the number of coarse clusters, c is the maximum number of non-duplicate documents in a cluster, and k_{max} is the maximum number of templates in a coarse-grained cluster.

Proof. We analyze the runtimes of INFOSHIELD-COARSE and INFOSHIELD-FINE separately. For INFOSHIELD-COARSE, we iterate through N documents, picking the top 10% of phrases in N, and adding edges between these documents and phrases. Thus the runtime of INFOSHIELD-COARSE is $O(Nl)$, where l is the average length of the documents.

In INFOSHIELD-FINE, there are a total of k iterations, where k is the maximum number of templates generated from the given data. With the help of vectorization, MSA can be done in $O(l^2)$. For each iteration, *Consensus-Search* requires $O(\log S^{'} \times S^{'} l^2)$ time, where $S^{'}$ is the average number of documents being aligned in each template; and *Slot-Detection* requires $S^{'} l^2$ time. The time complexity of *Candidate-Alignment* in each iteration is $O(Sl^2)$, where $S \geq S^{'}$ is the average number of documents in the each cluster. Thus the time complexity of INFOSHIELD-FINE is $O(\sum_{i=1}^{m} k_i S_i \log(S_i) l^2)$, which is upper-bounded by $O(k_{max} N \log(N) l^2)$, and where where m is the number of coarse clusters generated by INFOSHIELD-COARSE, k_{max} is the maximum number of templates generated by a cluster.

In total, the algorithm takes time $O(Nl) + O(k_{max} N log(N) l^2)$ time.

In practice, $k_{max} \leq 2$ in the Twitter datasets. Furthermore, the value of c will be quite low, since Twitter spambots post many duplicate tweets, which will make the runtime fast. Empirical evidence of this can be found in Figure 2, where we see that INFOSHIELD-COARSE scales linearly with input size. For the use cases presented in this paper, i.e. escort advertisements and tweets, we also note that l is bounded (280 for Tweets). ■

V. EXPERIMENTS

A. Description

Here, we give descriptions of all datasets and metrics, as well as the experimental setup.

1) Twitter Bot Data: We use data from [4]. This data includes the tweet text and user id. The data is split into the following categories:

Dataset	Accounts	Tweets
genuine accounts	3,474	8,377,522
social spambots #1	991	1,610,176
social spambots #3	464	1,418,626
Test set #1 (spambots #1)	1,982	4,061,598
Test set #2 (spambots #3)	928	2,628,181

TABLE VII: Statistics for Twitter bot data

To create each test set, [4] sampled all tweets from 50% genuine accounts, and 50% from either social spambots #1 or social spambots #3. We use the provided test sets, which focus on social spambots only, so we can easily compare results to the best performing methods in [4].

This data not only contains binary labels as to whether particular tweets were posted from bots or legitimate users, but also inherent clusters; i.e. user ids that correspond to legitimate users or bots.

We expect INFOSHIELD to cluster most tweets from bots in clusters, ideally in one cluster per bot, and to have few clusters with legitimate users in them. With this intuition, we can create ground truth cluster labels in Twitter data as follows: (1) all legitimate users get labeled -1, since we assume their tweets are different enough that they shouldn't be clustered together; (2) all bots get labeled with their user id.

2) Human Trafficking Data - Trafficking10k Dataset: The Trafficking 10k dataset is created in [7], where expert annotators manually labeled 10,265 ads from 0-6. 0 represents "Not Trafficking", 3 represents "Unsure", and 6 represents "Trafficking".There are 6,551 ads labeled as not HT, 354 labeled as "Unsure", and 3,360 labeled as HT.

Since the likelihood of an ad being HT is subjective, labeling is a difficult task. In fact, our analysis shows that 40% of exact duplicate ads (without any preprocessing) had label disagreement – i.e. multiple labels for the same exact text. Ads that are exact duplicates account for 12% of the dataset. We expect this labeling issue to occur for near duplicates as well. Therefore, we argue that looking at ads individually, whether

manually or algorithmically, is a non-ideal way to find or to label HT cases.

Despite the noisy labels, this is the only HT dataset to our knowledge with labeled data by human investigators. Thus, we use this dataset in our experiments, while being aware that noisy labels may impact results.

This data does not have ground truth clusters. However, to create binary labels, we can call scores 0-3 as not HT, and 4-6 as HT.

3) Human Trafficking Data – Cluster Trafficking: Cluster Trafficking is a new dataset provided by Marinus Analytics. This data contains cluster labels, provided by domain experts, for both HT and spam advertisements. We are given 6 spam clusters as well as ads from 96 massage parlors around the US. Cluster Trafficking consists of 157,258 ads, with 6,283 spam ads, 50,985 HT ads, and 99,990 normal ads.

4) Baselines: Most state-of-the-art methods for HT detection are not open-sourced. Instead, we compare against HTDN [7], which usese the same Trafficking10k dataset, and develop three baselines using state-of-the art text embedding methods Word2Vec [23], FastText[25], and Doc2Vec [24]. We train all models using 1 million escort advertisements from the web. Then, we cluster using HDBSCAN [27] with a minimum cluster size of 3. We call these methods Word2Vec-cl, Doc2Vec-cl, and FastText-cl.

On Twitter data, we compare to three supervised methods [45], [14], [46] and one unsupervised method [47]. These methods all use Twitter-specific features that our domain-independent INFOSHIELD does not use, such as number of mentions, favorites, retweets, posting time, etc. The unsupervised method is also not fully automatic, as a manually set threshold discerns spam from legitimate tweets, which they change for each dataset. Regardless, INFOSHIELD provides comparable results to these baselines.

5) Metrics: For Twitter data, we have both binary labels and ground truth cluster labels. To compare binary labels, we can report precision, recall, and F1 score. For cluster labels, we use Adjusted Rand Index (ARI) [48].

We calculate precision, recall and F1 by calling all documents that ended up in templates to be suspicious, and all other documents as not suspicious.

B. Results

Here we report experiments to answer the following questions.

Q1. *Practical*: How fast is INFOSHIELD, and how well does INFOSHIELD work?

Q2. *Interpretable*: How well does INFOSHIELD visualize clusters? Are there any interesting results with respect to the relative length metric?

Q3. *Robustness*: How much does INFOSHIELD-COARSE change as we consider longer n-grams?

Then, we report advantages and observations about INFOSHIELD.

C. Q1 – Practical

How scalable is INFOSHIELD? By using INFOSHIELD-COARSE to create coarse-grained clusters, and using the more expensive INFOSHIELD-FINE on smaller input sizes, we save time. We design an experiment on Twitter data by sampling Tweets the same way [4] did to create the test sets, and report the average runtime for each dataset out of five trials. The result is shown in Figure 2. Error bars were too small to be visible, so they were omitted.

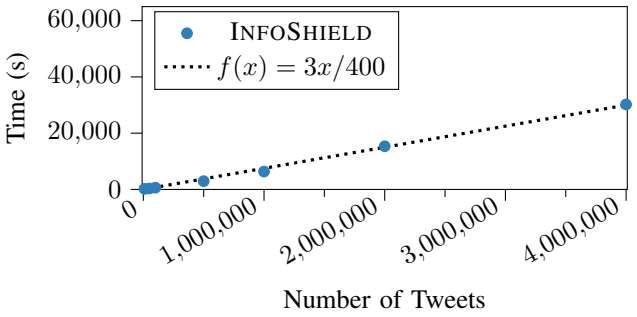


Fig. 2: INFOSHIELD *is scalable:* Linear on the input size; ≈8hours for 4M tweets, on a stock laptop.

How effective is INFOSHIELD? We run INFOSHIELD, as well as our developed baselines on both the Twitter data and Trafficking10k datasets. We report our results in Table VIII, comparing against the two highest performing methods from [4].

On Twitter data, INFOSHIELD always performs within ten points of the top contender, despite using no features specific to Twitter such as retweets, favorites, or posting times.

For HT data, we see that INFOSHIELD reports the highest precision; this is crucial since we want to avoid giving false positives to law enforcement at all costs. Law enforcement would rather know that they receive a real HT case (precision) than for all HT cases to be returned (recall) since they likely won't have time to pursue all cases. False positives cause law enforcement to lose trust in the algorithm and abandon it – as happened with previous applied solutions.

D. Q2 – Interpretable

How well does INFOSHIELD visually interpret the clusters and templates we find? We show a few results of templates for Twitter data, and a censored version for the HT data, with discussion.

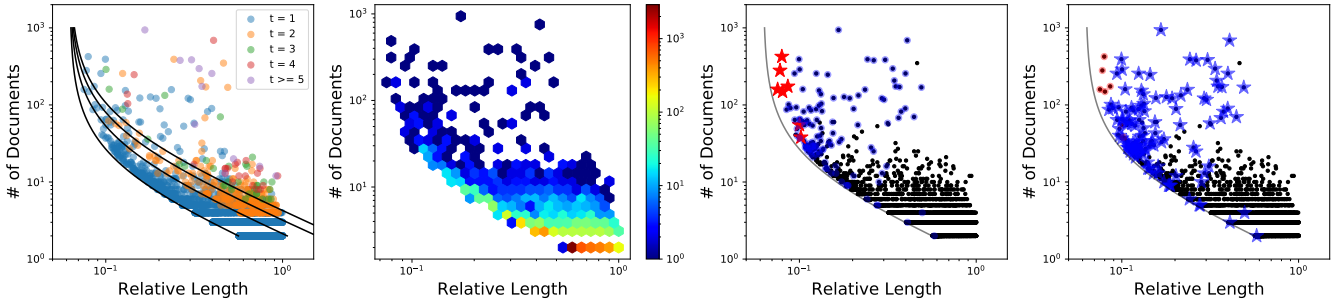
1) Twitter Data: As shown in Table IX, we find that 23 Spanish tweets are encoded by the given template. The first 22 ones are exact duplicates, but the last one contains three different words. INFOSHIELD-FINE automatically determines that representing those different terms as unmatched results, rather than as a slot, gives a smaller total cost. We can easily spot anomalies within clusters by using the template; the last tweet will have a lower compression rate than all other tweets.

In Table X, we find that all the tweets are talking about the most popular weekly stories. While the first half of all

Twitter Data								
Dataset	Test Set #1				Test Set #2			
Metric	ARI	Prec.	Rec.	F1	ARI	Prec.	Rec.	F1
INFOSHIELD	**83.2**	*93.0*	*91.2*	*92.1*	**75.7**	*96.7*	*88.9*	**92.6**
Cresci [47]	n/a	**98.2**	**97.2**	**97.7**	n/a	**100**	85.8	92.3
BotOrNot [14]	n/a	47.1	20.8	28.9	n/a	63.5	**95.0**	76.1
Yang [45]	n/a	56.3	17.0	26.1	n/a	72.7	40.9	52.4
Ahmed [46]	n/a	*94.5*	*94.4*	*94.4*	n/a	*91.3*	*93.5*	*92.3*

Human Trafficking Data							
Dataset	Trafficking10k			Cluster Trafficking			
Metric	Prec.	Rec.	F1	Prec.	Rec.	F1	ARI
INFOSHIELD	**84.8**	50.7	*63.5*	**85.4**	**99.8**	**92.0**	**43.1**
Word2Vec-cl	19.4	10.7	13.8	71.7	*99.5*	*83.1*	9.6
Doc2Vec-cl	25.6	10.9	15.3	74.2	*98.8*	*84.7*	16.2
FastText-cl	28.4	22.4	25.1	69.6	*99.6*	81.9	6.8
HTDN [7]	71.4	**62.2**	**66.5**	—	—	—	n/a

TABLE VIII: INFOSHIELD performs well: Notice that INFOSHIELD beats or approaches the best *domain-specific* method in both settings. Bold shows the best score, underline shows methods within 10 points of the best. Methods in red are supervised, while INFOSHIELD is unsupervised.



(a) Lower Bound (b) Heat Map of Clusters (c) Spam Clusters (red stars) (d) Human Trafficking Clusters (blue stars)

Fig. 3: *Perpetrators seem separable*, thanks to our features: (a) shows all clusters (circles) and the lower bounds (black lines) – points are above the lower bound, as expected. (b) heatmap of the same: most points are close to the lower bound. (c) emphasizes the spam clusters as red stars, and (d) emphasizes the HT clusters as blue stars. Note that the majority of spam and HT clusters (red and blue stars) sit apart from the benign clusters.

tweets are almost identical, with minor syntax differences, the second half describes the particular stories, which all differ. INFOSHIELD-FINE then detects the second half of each sentence as a slot, which we expect to have different content in each tweet. This will help researchers pay attention to the most worth-studying parts.

2) HT Data: In Table XI, we show an example template from the HT domain. Unfortunately, we must censor the text to protect potential HT victims, so we only provide the highlighting from the template. For the slots, we give a description of the type of text they represent.

Notice that slots tend to include consistent user-specific information. For example, the second slot, if not empty, always discusses time. With a quick glance, a domain expert can easily find this data, rather than looking at a longer wall of text. For the HT domain, interpretability is key: law enforcement will only have to read one template, rather than each cluster member individually, to determine if this cluster is suspicious.

The slots also contain messy data: i.e. while each slot has a specific purpose in Table XI, the text can be in multiple formats, i.e. "until 9pm" vs. "9 P.M", etc. Work could be done to automatically extract and process the information within each slot, but this is beyond the scope of this paper.

3) Relative Length: Next, we consider the relative length, to further investigate the clusters detected by INFOSHIELD. How does the relative length of a micro-cluster change as a function of the number of documents? Do we notice any differences between the relative lengths of spam clusters vs. HT clusters? Using the Cluster Trafficking dataset, we illustrate the lower bound of relative length versus number of documents per cluster in Figure 3(a), where the black lines from left to right denote the lower bound of clusters with one to four templates. For example, the clusters with two templates (orange dots) cannot be on the left side of the second black line. As shown in Figure 3(b), most clusters are concentrated by the lower bound, meaning that they do not have high numbers of documents. Further analysis surprisingly finds that spam and HT clusters follow patterns in this space. As shown in Figure 3(c), most spam clusters (red stars) have small relative length with a high number of documents; in Figure 3(d), there are two patterns of HT clusters (blue stars): (1) the near-duplicate clusters with a high number of documents (but slightly lower than spam clusters), (2) the outlier clusters that lie far from the lower bounds.

E. Q3 – Robustness

How sensitive is INFOSHIELD-COARSE to the length of n-grams we use to calculate tfidf scores? We run an experiment on one of the datasets we used for our timing experiments, which contains 100,000 tweets by sampling all tweets from 50% legitimate accounts and 25% social spambot #1 accounts, and 25% social spambot #3 accounts. We detail the results in Figure 4.

Constant Slot Insertion Deletion Substitution

T_1		**sismo**	**richter**	**km al sureste de puerto escondido oax lat lon pf km**
#1		sismo	richter	km al sureste de puerto escondido oax lat lon pf km
Omit	21 Identical	Tweets as #1 ...		
#23	sismologicomx	sismo	magnitud loc	km al sureste de puerto escondido oax lat lon pf km

TABLE IX: INFOSHIELD *is language-independent:* Spanish template from Twitter dataset.

Constant Slot Insertion Deletion Substitution

T_1	**the mostpopular**	**most popularstories on pr daily this week from**	*	**are**	*
#1	the	most popular stories on pr daily this week from	instagram to mr t and perhaps even your grocers produce httptcokbfwdfts		
...					
#14	the	most popular stories on pr daily this week from	new cover photo rules on facebook and a battle of the soci httptcoeuetyugbku		
#15	the mostpopular	stories on pr daily this week from	whimsical words to hillarys texts here	are	this weeks mos httptcoymwflapn
...					
#27	the mostpopular	stories on pr daily this week from	understanding sopa to dating a pr professional here	are	the httptcoploce

TABLE X: INFOSHIELD *detects slots:* template from Twitter dataset.

Constant Slot Insertion Deletion Substitution

T_1	**not shown for victim's safety**					
#1	(empty)	time	(empty)		(empty)	
#2	personal description	time	(empty)		cost	
#3	(empty)	time	(empty)		cost	
#4	personal description	(empty)	preferences		cost	
...18 similar ads						

TABLE XI: *Slots contain user-specific information:* template from HT dataset.

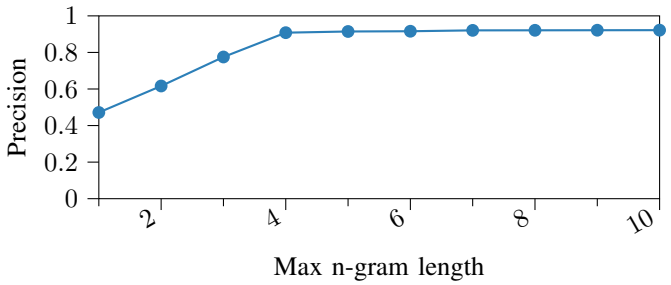


Fig. 4: *5-grams are enough:* Precision stabilizes after $n = 4$.

F. Discussion and Discoveries: INFOSHIELD *at Work*

We note that INFOSHIELD has the following advantages:

Advantage 1. INFOSHIELD *is* general*, using no language-specific or domain specific features.*

In fact, the Twitter data includes tweets in Spanish, Italian, English, and Japanese, and we use no language-specific features in our methodology. In INFOSHIELD-COARSE, we automatically let tf-idf penalize common words, so there is no need to include stop-words in our algorithm. Note that the template in Table IX is in Spanish, while the template in Table X is in English. This makes our method very powerful; it can be run on text in almost any language, or on other text data such as DNA strings.

Advantage 2. INFOSHIELD *is* extensible: *the goal of minimizing the total cost is separate from the algorithms we propose to do so.*

In fact, one could replace INFOSHIELD-COARSE and INFOSHIELD-FINE with similar algorithms achieving the same end goal of pre-clustering and minimizing the total cost. We propose the algorithms above because they are scalable, and effective on real data.

Advantage 3. INFOSHIELD *does not require any user-defined parameters.*

By using *Consensus-Search* to find the optimal algorithm, we remove the need for user-defined parameters in INFOSHIELD-FINE.

VI. CONCLUSIONS

We presented INFOSHIELD, which finds small clusters of near-duplicates in a collection of documents like escort ads for human trafficking detection, and visualizes the micro-clusters in a clear manner.

The main contributions of the method are that it is:

- *Practical*, processing *4 million* documents in *8 hours*, on a Dell Alienware laptop with 32GiB RAM and i7 processor, running Arch Linux,
- *Parameter-free & Principled*, based on the MDL principle,
- *Interpretable*, providing a clear visualization and summarization of clusters, and
- *Generalizable* and independent of domain (Twitter, HT), as well as of language (English, Spanish etc).

Reproducibility: Code is open-sourced here: https://github.com/mengchillee/InfoShield. The twitter datasets are public [4]. The Trafficking10K dataset is available after NDA – email Cara Jones (cara@marinusanalytics.com).

REFERENCES

[1] "Ilo global estimate of forced labour," http://www.ilo.org/wcmsp5/groups/public/---ed_norm/---declaration/documents/publication/wcms_182004.pdf, 2012. 1

[2] NA. Survivor survey r5. [Online]. Available: http://www.thorn.org/wp-content/uploads/2015/02/Survivor_Survey_r5.pdf 1

[3] ——. Human trafficking & online prostitution advertising. [Online]. Available: https://wagner.house.gov/Human%20Trafficking%20%26%20Online%20Prostitution%20Advertising 1

[4] S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, and M. Tesconi, "The paradigm-shift of social spambots: Evidence, theories, and tools for the arms race," in *WWW*, 2017, p. 963–972. 2, 8, 9, 11

[5] M. Kejriwal, J. Ding, R. Shao, A. Kumar, and P. A. Szekely, "Flagit: A system for minimally supervised human trafficking indicator mining," *CoRR*, vol. abs/1712.03086, 2017. 2

[6] H. Alvari, P. Shakarian, and J. E. K. Snyder, "Semi-supervised learning for detecting human trafficking," *Security Informatics*, vol. 6, no. 1, p. 1, 2017. 2

[7] E. Tong, A. Zadeh, C. Jones, and L. Morency, "Combating human trafficking with multimodal deep models," in *ACL*, Vancouver, Canada, 2017, pp. 1547–1556. 2, 3, 8, 9, 10

[8] S. S. Esfahani, M. J. Cafarella, M. B. Pouyan, G. J. DeAngelo, E. Eneva, and A. E. Fano, "Context-specific language modeling for human trafficking detection from online advertisements," in *ACL (1)*. Association for Computational Linguistics, 2019, pp. 1180–1184. 2

[9] L. Wang, E. Laber, Y. Saanchi, and S. Caltagirone, "Sex trafficking detection with ordinal regression neural networks," *arXiv preprint arXiv:1908.05434*, 2019. 2

[10] L. Li, O. Simek, A. Lai, M. Daggett, C. K. Dagli, and C. Jones, "Detection and characterization of human trafficking networks using unsupervised scalable text template matching," in *IEEE Big Data*, 2018, pp. 3111–3120. 2, 3

[11] K. Sharma, F. Qian, H. Jiang, N. Ruchansky, M. Zhang, and Y. Liu, "Combating fake news: A survey on identification and mitigation techniques," *ACM TIST*, vol. 10, no. 3, pp. 1–42, 2019. 2

[12] X. Zhou and R. Zafarani, "Fake news: A survey of research, detection methods, and opportunities," *arXiv preprint arXiv:1812.00315*, 2018. 2

[13] N. Shah, H. Lamba, A. Beutel, and C. Faloutsos, "The many faces of link fraud," 2017. 2

[14] C. A. Davis, O. Varol, E. Ferrara, A. Flammini, and F. Menczer, "Botornot: A system to evaluate social bots," in *WWW*, 2016, p. 273–274. 2, 9, 10

[15] N. Gleicher, *How We Respond to Inauthentic Behavior on Our Platforms: Policy Update*, 2019. [Online]. Available: https://about.fb.com/news/2019/10/inauthentic-behavior-policy-update/ 2

[16] M. Giatsoglou, D. Chatzakou, N. Shah, A. Beutel, C. Faloutsos, and A. Vakali, "Nd-sync: Detecting synchronized fraud activities," in *PAKDD (2)*, vol. 9078, 2015, pp. 201–214. 2

[17] Z. Harris, "Distributional structure," *Word*, vol. 10, no. 23, pp. 146–162, 1954. 3

[18] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *J. Documentation*, vol. 60, no. 5, pp. 493–502, 2004. 3

[19] V. Kanjirangat and D. Gupta, "A study on extrinsic text plagiarism detection techniques and tools," *Journal of Engineering Science and Technology Review*, vol. 9, pp. 150–164, 10 2016. 3

[20] V. Martins, D. Fonte, P. Rangel Henriques, and D. da Cruz, "Plagiarism detection: A tool survey and comparison," *OpenAccess Series in Informatics*, vol. 38, pp. 143–158, 01 2014. 3

[21] S. Elmanarelbouanani and I. Kassou, "Authorship analysis studies: A survey," *Int. Journal of Computer Applications*, vol. 86, 12 2013. 3

[22] S. Brin, J. Davis, and H. García-Molina, "Copy detection mechanisms for digital documents," in *SIGMOD*, 1995, p. 398–409. 3

[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *ICLR*, 2013. 3, 9

[24] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *ICML*, 2014, p. II–1188–II–1196. 3, 9

[25] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 135–146, 2017. 3, 9

[26] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *KDD*, vol. 96, no. 34, 1996, pp. 226–231. 3

[27] L. McInnes, J. Healy, and S. Astels, "hdbscan: Hierarchical density based clustering," *J. Open Source Softw.*, vol. 2, no. 11, p. 205, 2017. 3, 9

[28] M. Ankerst, M. M. Breunig, H. Kriegel, and J. Sander, "OPTICS: ordering points to identify the clustering structure," in *SIGMOD*, 1999, pp. 49–60. 3

[29] C. H. Q. Ding and X. He, "Principal component analysis and effective k-means clustering," in *SIAM DM*, 2004, pp. 497–501. 3

[30] M.-L. Lo and C. V. Ravishankar, "Spatial Joins Using Seeded Trees," *SIGMOD*, pp. 209–220, May 24-27 1994. 3

[31] T. Brinkhoff, H.-P. Kriegel, and B. Seeger, "Efficient Processing of Spatial Joins Using R-Trees," in *SIGMOD*, 1993, pp. 237–246. 3

[32] A. Guttman, "R-Tree : A dynamic Index Structure for Spatial Searching," in *SIGMOD*, Boston, MA, 1984, pp. 47–57. 3

[33] G. J. Barton and M. J. Sternberg, "A strategy for the rapid multiple alignment of protein sequences: confidence levels from tertiary structure comparisons," *Journal of molecular biology*, vol. 198, no. 2, pp. 327–337, 1987. 3

[34] C. Lee, C. Grasso, and M. F. Sharlow, "Multiple sequence alignment using partial order graphs," *Bioinformatics*, vol. 18, no. 3, pp. 452–464, 2002. 3, 6

[35] R. Barzilay and L. Lee, "Learning to paraphrase: An unsupervised approach using multiple-sequence alignment," in *HLT-NAACL*, 2003. 3

[36] S. Shen, D. R. Radev, A. Patel, and G. Erkan, "Adding syntax to dynamic programming for aligning comparable texts for the generation of paraphrases," in *COLING/ACL*, 2006, pp. 747–754. 3

[37] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, no. 5, p. 465–471, Sep. 1978. 3

[38] P. Grünwald, *The Minimum Description Length Principle*. MIT Press, 2007. 3

[39] M. Mehta, R. Agrawal, and J. Rissanen, "SLIQ: A fast scalable classifier for data mining," in *EDBT*, Mar. 1996. 3

[40] D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos, "Fully automatic Cross-associations," in *KDD*, 2004. 3

[41] Y. Matsubara, Y. Sakurai, and C. Faloutsos, "AutoPlait: automatic mining of co-evolving time sequences," in *SIGMOD*, 2014, pp. 193–204. 3

[42] E. Keogh, S. Lonardi, and C. A. Ratanamahatana, "Towards Parameter-Free Data Mining," in *KDD*, 2004. 3

[43] J. Rissanen, "A Universal Prior for Integers and Estimation by Minimum Description Length," *Annals of Statistics*, vol. 11, no. 2, pp. 416–431, 1983. 5

[44] E. K. Chong and S. H. Zak, *An introduction to optimization*. John Wiley & Sons, 2004. 6

[45] C. Yang, R. C. Harkreader, and G. Gu, "Empirical evaluation and new design for fighting evolving twitter spammers," *IEEE Trans. Inf. Forensics Secur.*, vol. 8, no. 8, pp. 1280–1293, 2013. 9, 10

[46] F. Ahmed and M. Abulaish, "A generic statistical approach for spam detection in online social networks," *Comput. Commun.*, vol. 36, no. 10-11, pp. 1120–1129, 2013. 9, 10

[47] S. Cresci, R. D. Pietro, M. Petrocchi, A. Spognardi, and M. Tesconi, "Dna-inspired online behavioral modeling and its application to spambot detection," *IEEE Intell. Syst.*, vol. 31, no. 5, pp. 58–64, 2016. 9, 10

[48] L. Hubert and P. Arabie, "Comparing partitions," *Journal of classification*, vol. 2, no. 1, pp. 193–218, 1985. 9