DeltaShield: Information Theory for Human- Trafficking Detection

CATALINA VAJIAC and MENG-CHIEH LEE, Carnegie Mellon University
AAYUSHI KULSHRESTHA and SACHA LEVY, McGill University & Mila
NAMYONG PARK, Carnegie Mellon University
ANDREAS OLLIGSCHLAEGER and CARA JONES, Marinus Analytics
REIHANEH RABBANY, McGill University & Mila
CHRISTOS FALOUTSOS, Carnegie Mellon University

Given a million escort advertisements, how can we spot near-duplicates? Such micro-clusters of ads are usually signals of human trafficking (HT). How can we summarize them to convince law enforcement to act? 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: *practical*, being scalable and effective on real data; *parameter-free and principled*, requiring no user-defined parameters; *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 *generalizable*, beating or matching domain-specific methods in Twitter bot detection and HT detection, respectively, as well as being language independent. Interpretability is particularly important for the anti-HT 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 HT ads with 84% precision. Moreover, it is scalable, requiring about 8 hours for 4 million documents on a stock laptop. Our incremental version, DeltaShield, allows for fast, incremental updates, with minor loss of accuracy.

CCS Concepts: • **Computing methodologies** → **Anomaly detection**; • **Information systems** → **Clustering;**

Additional Key Words and Phrases: Text mining, minimum description length (MDL), anti-human trafficking detection

ACM Reference format:
Catalina Vajiac, Meng-Chieh Lee, Aayushi Kulshrestha, Sacha Levy, Namyong Park, Andreas Olligschlaeger, Cara Jones, Reihaneh Rabbany, and Christos Faloutsos. 2023. DeltaShield: Information Theory for Human-Trafficking Detection. *ACM Trans. Knowl. Discov. Data.* 17, 2, Article 28 (March 2023), 27 pages.
https://doi.org/10.1145/3563040

This material was based on work supported by the National Science Foundation Graduate Research Fellowship under grants DGE1745016 and DGE2140739.
Authors' addresses: C. Vajiac, M.-C. Lee, N. Park, and C. Faloutsos, Carnegie Mellon University; emails: {cvajiac, mengchil, namyongp, christos}@cs.cmu.edu; A. Kulshrestha, S. Levy, and R. Rabbany, McGill University & Mila; emails: {aayushi.kulshrestha, sacha.levy}@mail.mcgill.ca, rrabba@cs.mcgill.ca; A. Olligschlaeger and C. Jones, Marinus Analytics; emails: {olli, cara}@marinusanalytics.com.

1556-4681/2023/03-ART28 $15.00
https://doi.org/10.1145/3563040

1 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 (HT)** 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-duplication, and more.

In this article, 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).

1.1 Application to the HT Domain

Although InfoShield is general, our main motivation is near-duplicate detection and summarization in escort advertisements. HT is a dangerous societal problem that 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 [25]. The majority of victims are advertised online [42], and 56% of victims have no input on ad content [42]. The average pimp has four to six victims [41]. Thus, the majority of ads suspected of HT are written by one person, who is controlling ads for four to six different victims at a time. By looking for small clusters of ads that contain similar phrasing rather than analyzing stand-alone ads, we are 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.

1.2 Application to Twitter Bot Detection

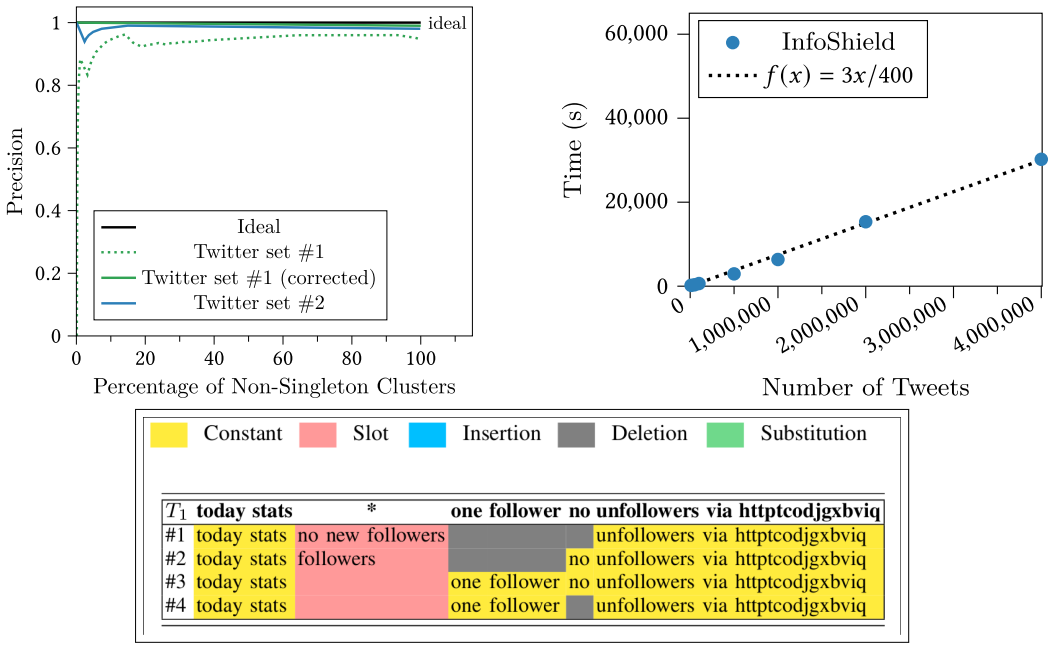
Detection of organized activity also has a clear application to bot detection; given millions of tweets, most of which come from legitimate users, how can we find tweets that exhibit bot-like behavior? The simplest kind of bot behavior is spamming (i.e., posting tweets that are almost or exactly identical 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, such as the number of retweets [12, 13]. 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.

1.3 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 (bottom). 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 preprocessing method, InfoShield-coarse, which dramatically improves scalability to be quasi-linear, by (1) eliminating single-copy documents/ads and (2) grouping the rest in coarse, but mainly homogeneous, clusters.

The resulting algorithms, InfoShield and DeltaShield, have a long list of desirable properties, such as the following:



T_1	today stats	*	one follower	no unfollowers via httptcodjgxbviq
#1	today stats	no new followers		unfollowers via httptcodjgxbviq
#2	today stats	followers		no unfollowers via httptcodjgxbviq
#3	today stats		one follower	no unfollowers via httptcodjgxbviq
#4	today stats		one follower	unfollowers via httptcodjgxbviq

Fig. 1. InfoShield works. (Top left) Precision@k on Twitter data is close to ideal. (Top right) The scalability of InfoShield over different data sizes. (Bottom) The interpretability of InfoShield, detecting and visualizing slots (in red) (i.e., portions of tweets that highly differ between otherwise duplicate documents).

- *Practical*, being scalable and 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*, and we show results on two diverse areas, namely Twitter data and HT data, as well as on multiple languages (i.e., Spanish, Italian, and English).
- *Incremental*, processing new batches of documents on-the-fly without recomputing on historical documents.

A system diagram explaining the pipeline of InfoShield and DeltaShield is shown in Figure 2.

Reproducibility. Our code is open sourced at https://github.com/catvajiac/InfoShield-Incremental. The HT dataset is available to researchers after NDA (email Cara Jones at cara@marinusanalytics.com). The Twitter datasets are publicly available (see [11]).

2 BACKGROUND AND RELATED WORK

There is a lot of work on HT detection, document clustering, and **Multiple-Sequence Alignment (MSA)**, and we group it in the following sections.

2.1 HT Detection

Some previous works try to classify whether or not a particular advertisement is suspected of HT [2, 17, 28, 52]. For instance, HTDN [52] proposes a supervised deep multimodal model trained on 10K manually labeled ads. Unfortunately, due to the adversarial nature of escort advertisements, these predefined or learned features do not 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

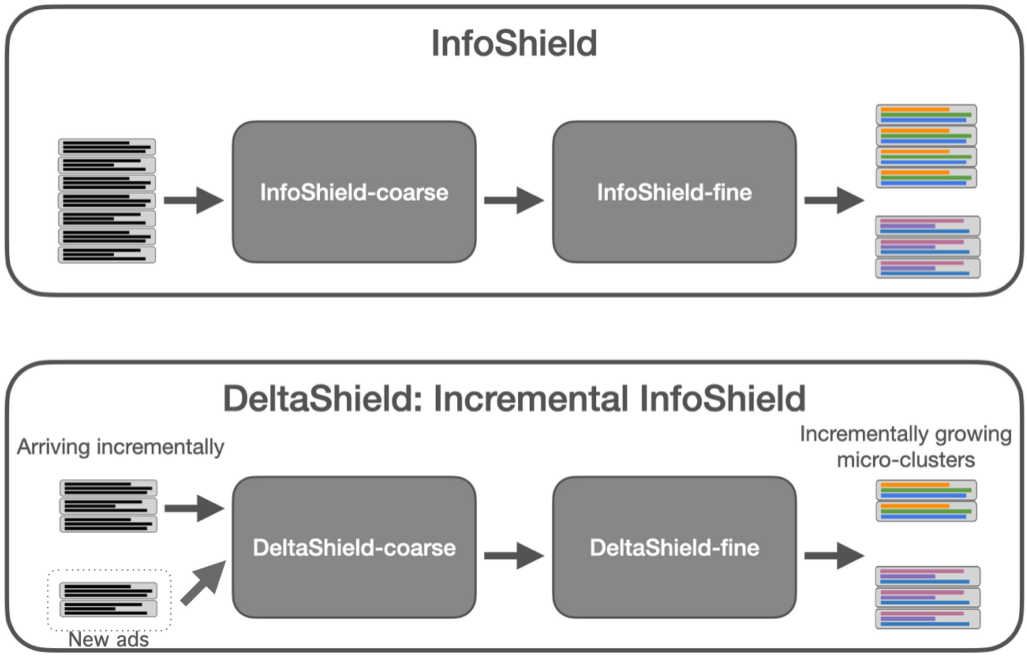


Fig. 2. A system diagram of InfoShield and DeltaShield, showing the input, output, and intermediate steps.

discussed in Section 6. 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 connections between ads [43–45] and *groups* of organized activity are preferred in this domain [34]. In particular, Template Matching [34] proposes the first anti-HT method to our knowledge to perform clustering. However, the interpretability of clusters is limited, and the algorithm is not scalable.

2.2 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 [50, 54]. Fewer works look for anomalies or fraud in networks rather than in text, for instance [49]. A notable method, Botometer [13], 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 6. Cresci et al. [11] give a more comprehensive overview of Twitter bot detection methods, and also provide the dataset we will use in Section 6.1.1. 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 [20]. ND-Sync [19] finds a related but different type of behavior—that is, "retweet spam"—where groups of multiple users exhibit organized behavior by consistently upvoting a particular user's tweets.

2.3 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 [23] and **Term Frequency–Inverse Document Frequency (tf-idf)** [26]. These methods are commonly used for plagiarism detection [7, 16, 27, 36], which is a similar setting to near-duplicate detection. However, none of

Table 1. InfoShield Matches All Specs, Whereas Competitors Miss One or More of the Features

Property \ Method	Clustering [3, 15, 18, 38]	Barzilay and Lee [5]	Shen et al. [51]	HDTN [52]	Template Matching [34]	InfoShield	DeltaShield
Practical: Scalable	?			✓	✓	✓	✓
Practical: Effective	?	?	?	?	✓	✓	✓
Practical: Ranked output	?			?	✓	✓	✓
Parameter-free	?					✓	✓
Principled						✓	✓
Interpretable	?	✓	✓		✓	✓	✓
Slot Detection		✓				✓	✓
Generalizable	✓					✓	✓
Incremental							✓

A question mark (?) means that it depends on the specific clustering method, or that it is unclear from the original paper.

these methods do visualization or ranking, and some assumptions do not work in our case. For example, Brin et al. [7] assume that 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 [40], Doc2Vec [31], and FastText [6] 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 6.

BERT [14] is another successful language model, but through experiments on the Trafficking10k dataset, we find it does not perform well on escort ad text [30], due to the sheer number of misspellings, shortenings, and specific escort keywords not found in normal text. Instead, we take the approach of developing a lighter-weight solution that naturally handles the small amount of labeled data.

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 [18], HDBSCAN [38], OPTICS [3], or k-means [15]. These are all powerful methods, but none of them do slot detection. We compare InfoShield to HDBSCAN as part of our curated baseline for HT detection (see Section 6 for more details).

In Table 1, we give several question marks for clustering methods because some of the methods are scalable (k-means), whereas 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" [8, 35]. However, these methods are best for low-dimensional spaces, since they use the R-tree [22] spatial access method.

2.4 Multiple-Sequence Alignment

MSA is a well-studied area with an application to biology, for comparing DNA sequences. The Barton-Sternberg algorithm [4] is an early profile-based approach that aligns sequences by updating a profile sequence iteratively. However, profile-based approaches generate ambiguity among sequences. To solve this, Lee et al. [32] use partial order graphs instead of profile sequences, which enables a base in dynamic programming to have multiple predecessors and successors.

Natural language processing is another area benefiting from MSA. Barzilay and Lee [5] apply MSA to learn the patterns of given word sequences by word lattices and rewrite the sentences. Shen et al. [51] focus 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.

2.5 Minimum Description Length

The MDL principle [46] 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 $C(D|M)$, whereas several other methods ignore the model complexity.

MDL has been extremely successful in several data mining tasks [21], including decision trees (SLIQ [39]), graph mining (CrossAssociations [9]), time series segmentation and mining (AutoPlait [37]), string similarity [29], 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.

Although all of the preceding methods have provided unique and interesting contributions, none have all of the same features as InfoShield. Table 1 contrasts InfoShield against the state-of-the-art competitors. The algorithms in Sections 3 and 4 appeared in the conference version of this work [33]. In this journal version, we have added the incremental algorithm, DeltaShield (Section 5), and more experiments (Section 6).

3 PROPOSED METHOD: THEORY

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

3.1 Intuition and Theory

Our problem is split into the following two 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?
(2) *Algorithms*: How do we quickly find clusters that describe patterns in the data concisely (InfoShield-coarse: Section 4.1) and then how do we refine and visualize these clusters (InfoShield-fine: Section 4.2).

Our MDL-based approach is best explained with examples.

Example 1 (Simple Toy Example). Suppose we have the documents of Table 2 in a particular cluster. These documents are a shortened version of escort ads InfoShield clustered into one template.

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

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

Table 2. Simple Toy Example

Doc	Text
#1	Hi gentlemen, Korea super model just arrived...Alma and Joan specially selected...
#2	Hi gentlemen, Korea super model just arrived...Paula and Miya specially selected...
#3	Hi gentlemen, Korean super model just arrived...Paula specially selected...

Table 3. Full Toy Example

Doc	Text
#4	Gentlemen, Korea super model just arrived...Miya is specially selected...
#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	Hello, Anna here! My hours are...

Table 4. Templates for the Full Toy Example

Constant Slot Insertion Deletion Substitution

T_1	**Hi**	**gentlemen,**	**Korea**	**super model**	**just arrived...**	*	**specially selected...**
#1	Hi	gentlemen,	Korea	super model	just arrived...	Alma and Joan	specially selected...
#2	Hi	gentlemen,	Korea	super model	just arrived...	Paula and Miya	specially selected...
#3	Hi	gentlemen,	Korean	super model	just arrived...	Paula	specially selected...
#4		Gentlemen,	Korea	super model	just arrived...	Miya	specially selected...

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

preceding three-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 4.

Let us also consider an example of 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 in Table 3.

Doc #4 belongs in T_1, but with one deletion (omitting "a"), one insertion (adding "so"), and one substitution (replacing "great" with "good"). However, Docs #5 to #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 Docs #1 to #4, T_2 representing Docs #5 and #6, and Doc #7 does not belong to any template.

Furthermore, we would like to visualize the templates we do find. 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: "Hi gentlemen, Korea super model just arrived... * specially selected...
T_2: "I made 30K working * - call * or visit *"

(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 the tokens in slots, position and token for insertions, position for deletions, position and token

Table 5. Example Encoding for $C(D|M)$

Doc	Temp.	Slots	Ins.	Del.	Sub.
#1	T_1	{"Alma and Joan"}			
#2	T_1	{"Paula and Miya"}			
#3	T_1	{"Paula"}			3: "Korean"
#4	T_1	{"Miya"}		1	
#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"			

for substitutions, and the template id that best matches the document. Table 5 shows the information we include in $C(D|M)$ for our running example.

Notice that Docs #1 to #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) shown previously. More formally, we have the following definition.

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 3.2, we explain the exact cost for N documents and t templates more precisely. Then, in Section 4, 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 4 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.

3.2 Data Compression and Summarization

In this section, we give the details of the encoding cost in InfoShield. Table 6 provides symbols and definitions relevant to the encoding.

3.2.1 Template Encoding. We use the notation $\langle n \rangle$ for the coding cost of integer n, using the universal code length [47]—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 in the total cost $\langle l \rangle + l * \lg V$.

Definition 2 (Model Encoding Cost). The coding cost for t templates is given by

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

Let us describe every term of the preceding definition:

- $\langle t \rangle$: Universal coding, for the number of templates T.
- For each template T_i, we need

Table 6. Symbols and Definitions for InfoShield-fine

Symbol	Definition
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 the 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 $(1 \leq i \leq L)$

— $\langle l_i \rangle$ 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

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

3.2.2 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

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

where D_i denotes the data encoded by template T_i. We describe this definition in more detail. Let D_U denote the documents that do not match any template. The encoding cost for data $d \in D_U$ that 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").
- $\langle \hat{l}_d \rangle$ 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:

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

- repeat, for all other editing operations.

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

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

3.2.3 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.

4 PROPOSED METHOD: ALGORITHMS

How can we find templates that minimize our cost function in a scalable way? Although the intuition described in Section 3 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.

4.1 InfoShield-coarse

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

4.1.1 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 n-grams, with $n = 5$.[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 prevent InfoShield-coarse from being domain specific.

4.1.2 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 (due to an "unimportant" phrase that combined documents that ideally should not be combined), we rely on InfoShield-fine to refine these

[1]Phrase length has little impact on results past $n = 5$: see Section 6.5.

ALGORITHM 1: 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) ;

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.

4.2 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 have 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 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.

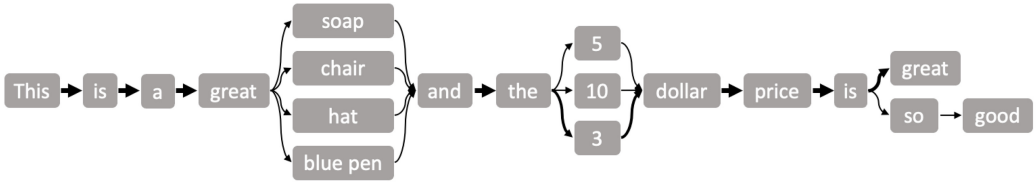
Let us take the first template from Table 4 as an example. We show a visual representation of what each step does in Figure 3.

To compute the MSA, we carefully choose to use **Partial Order Alignment (POA)** [32] 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.

4.2.1 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.

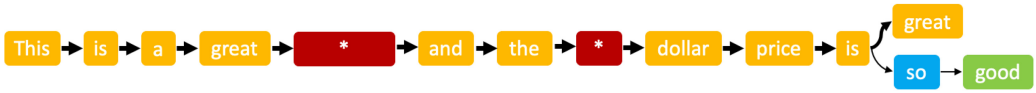
4.2.2 Consensus Search. After generating alignment A_i, how do we decide which tokens are part of the template, and which are insertions/deletions/substitutions? Keeping too many words in the template causes more unmatched operations (insertion/deletion/substitution), whereas keeping too few words hurts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



(a) Step 1: Candidate Alignment

(b) Step 2: Consensus Search



(c) Step 3: Slot Detection

Fig. 3. Example pipeline of InfoShield-fine. Here we show the output after each step of InfoShield-fine.

ALGORITHM 2: *Consensus-Search*

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 $T_i^{'}$;

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 one-dimensional, being relatively easy to solve. Hence, we employ the Dichotomous Search algorithm [10] 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.

4.2.3 Slot Detection. Once we have a template, how do we find slots? Slots contain parts of documents that we expect 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

ALGORITHM 3: *Slot-Detection*

```
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 ∈ A_i do
    x = 0;
    for j = 1, ..., l_a do
        if a_j is an insert or substitution word then
            P[x] ← P[x] + 1;
        else
            /* a_j is a matched or deleted word */
            x ← x + 1;
for p ∈ P do
    if C(D_i|T_i'(p.slot ← True)) < C(D_i|T_i') then
        T_i ← T_i(p.slot ← True);
Return T_i;
```

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.

4.2.4 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

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

ALGORITHM 4: InfoShield-fine

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 $\mathcal{T}$;

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 $\frac{t}{n} + \frac{1}{\lg V}$. □

4.2.5 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.

4.3 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 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 4, where we see that InfoShield-coarse scales linearly with input size. For the use cases presented in this article (i.e., escort advertisements and tweets), we also note that l is bounded (280 for tweets). □

5 PROPOSED METHOD: INCREMENTAL

To do HT detection in practice, we must develop an algorithm that can process documents incrementally. Domain experts have hundreds of millions of ads and keep crawling additional ones each day. If we have already grouped historical ads into t clusters, we want to process a batch of newly crawled documents without recomputing on historical documents. Here we will discuss the necessary modifications to InfoShield and present DeltaShield, with relevant experiments in Section 6.

5.1 DeltaShield-coarse

How can we modify InfoShield-coarse to be conducive to an online setting? We consider a setting where batches of documents come in during an aggregated time period (i.e., daily or weekly). Most of the algorithm can be adopted with minimal changes; since InfoShield-coarse incrementally adds to the document-term graph, we can process an entire batch, send the results to InfoShield-fine, and then continue processing documents as they arrive. The biggest challenge we have is in computing the tf-idf score of n-grams in a given document before seeing the entire corpus. To this end, we approximate the tf-idf score by computing the idf only on *the documents seen so far* rather than the entire corpus. This approach is advantageous for two reasons: (1) as we process more and more documents, the approximate tf-idf score of a given n-gram will approach its actual tf-idf score, as verified empirically in Section 6.6, and (2) for the HT application, domain experts have a lot of historical, inactionable data that can be processed first to improve the approximate tf-idf score.

5.2 DeltaShield-fine

In an online setting, we still need to generate new templates if needed, so Algorithm 4 will be performed in every batch. Moreover, a preprocessing step and an updating step must be included to keep DeltaShield efficient and effective.

5.2.1 Preprocess. We propose Algorithm 5 as a preprocessing step before trying to generate a new template in Algorithm 4. If we were able to process all documents at once, the intuitive

ALGORITHM 5: *Preprocess-Naive*

Data: An incoming document d_1, and a template set $\mathcal{T}$
Result: An updated template set $\mathcal{T}$ or False if no appropriate template
/* Examine all the templates */
$i^* = \arg\min_{T_i \in \mathcal{T}} C(d_1|T_i)$;
if $C(d_1|T_{i^*}) < C(d_1)$ **then**
  $D_{i^*} \leftarrow D_{i^*} \cup d_i$;
  $T_{i^*}.added \leftarrow True$;
  /* Find appropriate template -> Continue with the next incoming document */
  Return $\mathcal{T}$;
/* No appropriate template -> Used as an initial document to generate the new template */
Return False;

ALGORITHM 6: *Preprocess-ES*

Data: An incoming document d_1, and a template set $\mathcal{T}$
Result: An updated template set $\mathcal{T}$ or False if no appropriate template
$I_i \leftarrow |Intersection(d_1, T_i)|$ for all $T_i \in \mathcal{T}$;
$\mathcal{T}^* \leftarrow \mathcal{T}$ ordered by I;
for $T_{i^*} \in \mathcal{T}^*$ **do**
  /* Early stopping */
  if $C(d_1|T_{i^*}) < C(d_1)$ **then**
    $D_{i^*} \leftarrow D_{i^*} \cup d_i$;
    $T_{i^*}.added \leftarrow True$;
    /* Find appropriate template -> Continue with the next incoming document */
    Return $\mathcal{T}$;
/* No appropriate template -> Used as an initial document to generate the new template */
Return False;

solution is to go through all the documents and generate templates. However, in an online setting, we often see documents from any one template span over multiple batches. To this end, the preprocessing step tries to encode an incoming document by all existing templates in its coarse cluster and select the template with the lowest encoding cost. If the cost by the selected template is lower than the encoding cost of the document itself, we consider that the document belongs to that template.

Unfortunately, the time complexity of examining all the existing templates is $O(k_{max}l^2)$. If a coarse cluster has a large number of templates, we will incur a large overhead. To address this, we adopt an **Early-Stopping (ES)** mechanism in Algorithm 6. Instead of sequentially investigating all templates in a coarse cluster, we order the templates by the lengths of intersection between unigrams in the incoming document and each template. Then, we select the first template that lowers the encoding cost of the document.

Lemma 3. *The time complexity of naive preprocessing step is*

$$O(k_{max}l^2) \tag{11}$$

but can be reduced by the ES mechanism to

$$O(l^2 + k_{max}l), \tag{12}$$

where l is the (maximum) length of a document and k_{max} is the maximum number of templates in a coarse-grained cluster.

ALGORITHM 7: *Template-Update*

Data: A template set $\mathcal{T}$ and data D
Result: An updated template set $\mathcal{T}$
for $T_i \in \mathcal{T}$ **do**
  if $T_i.added$ **then**
    $A_i \leftarrow MSA(A_i, D_i)$;
    $T_i^{'} \leftarrow ConsensusSearch(A_i, D_i)$;
    $T_i^{*} \leftarrow SlotDetection(T_i^{'}, A_i, D_i)$;
    if $C(D_i|T_i^{*}) < C(D_i|T_i)$ **then**
      $T_i \leftarrow T_i^{*}$;
Return $\mathcal{T}$;

Proof. To compute the cost after compression, we calculate the alignment, which takes $O(l^2)$. To search for the template with the lowest encoding length, we examine $O(k_{max})$ templates. In total, the naive preprocessing step takes $O(k_{max}l^2)$ time to find the template with the lowest encoding cost.

Next we analyze the ES mechanism. The time complexity of computing the lengths of intersection for one template is $O(l)$. It takes $O(k_{max}l)$ to compute all the lengths for all templates in the coarse cluster. The total time complexity is then reduced to $cl^2 + k_{max}l$, where c denotes the number of templates examined. However, c is a small number in most cases (close to 1), which is negligible, so the final time complexity is $l^2 + k_{max}l$. □

Later in Section 6.6.2, we will demonstrate that the ES mechanism largely improves the efficiency while achieving comparable effectiveness.

5.2.2 Template Update. Once the new documents are added into a template, its representation will be slightly changed. It is also important to update the template to represent new documents. Hence, we perform an updating step, Algorithm 7, right after Algorithm 4. It is worth noting that we only update templates that now represent any new documents. Furthermore, since changes in templates tend to be gradual over time, it is not necessary to process the updating step in every batch. We can set either a threshold (e.g., 200 more documents) or an interval (e.g., 1 month) to trigger this step to improve the efficiency. We will demonstrate the tradeoff between effectiveness and scalability using interval and threshold setting in Section 6.6.2.

6 EXPERIMENTS

6.1 Description

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

6.1.1 Twitter Bot Data. We use data from [11] (Table 7). This data includes the tweet text and user id. The data is split into the following categories.

To create each test set, Cresci et al. [11] 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 their work [11].

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

Table 7. Statistics for Twitter Bot Data

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

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 should not be clustered together; (2) all bots get labeled with their user id.

6.1.2 HT Data: Trafficking10k Dataset. The Trafficking10k dataset is created in the work of Tong et al. [52], where expert annotators manually labeled 10,265 ads from 0 to 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 from 0 to 3 as not HT and those from 4 to 6 as HT.

6.1.3 HT 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 for a strange behavior, which we will call *escort spam*.

Definition 4 (Escort Spam). Escort spam is script-generated advertisements that do not actually advertise real escort workers. The purpose of escort spam is not known, but it serves to confuse law enforcement.

We are given six spam clusters as well as ads from 96 massage parlors around the United States. Cluster Trafficking consists of 157,258 ads, with 6,283 spam ads, 50,985 HT ads, and 99,990 normal ads.

6.1.4 Baselines. Most state-of-the-art methods for HT detection are not open sourced. Instead, we compare against HTDN [52], which uses the same Trafficking10k dataset, and develop three baselines using state-of-the art text embedding methods Word2Vec [40], FastText[6], and Doc2Vec [31]. We train all models using 1 million escort advertisements from the web. Then, we cluster using HDBSCAN [38] 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 [1, 13, 53] and one unsupervised method [12]. These methods all use Twitter-specific features that our domain-independent

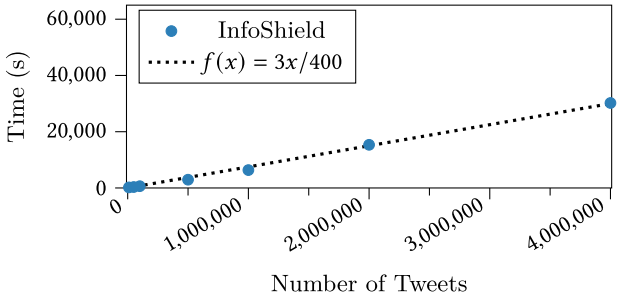


Fig. 4. InfoShield is scalable. Linear on the input size; ≈8 hours for 4 million tweets, on a stock laptop.

InfoShield does not use, such as number of mentions, favorites, retweets, and posting time. 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.

6.1.5 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 the **Adjusted Rand Index (ARI)** [24].

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.

6.2 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.** *Robust*: How much does InfoShield-coarse change as we consider longer n-grams?
- **Q4.** *Incremental*: How does DeltaShield compare with InfoShield in terms of efficiency and effectiveness?

Then, we report advantages and observations about InfoShield.

6.3 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 Cresci et al. [11] did to create the test sets, and report the average runtime for each dataset out of five trials. The result is shown in Figure 4. Error bars were too small to be visible, so they were omitted.

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 8, comparing against the two highest-performing methods from Cresci et al. [11].

On Twitter data, InfoShield always performs within 10 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

Table 8. InfoShield Performs Well: Notice That InfoShield Beats or Approaches the Best *Domain-Specific* Method in Both Settings

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 [12]	N/A	**98.2**	**97.2**	**97.7**	N/A	**100**	85.8	92.3
BotOrNot [13]	N/A	47.1	20.8	28.9	N/A	63.5	**95.0**	76.1
Yang et al. [53]	N/A	56.3	17.0	26.1	N/A	72.7	40.9	52.4
Ahmed & Abulaish [1]	N/A	*94.5*	*94.4*	*94.4*	N/A	*91.3*	*93.5*	*92.3*

HT 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 [52]	71.4	**62.2**	**66.5**	—	—	—	N/A

Bold shows the best score, and an underline shows methods within 10 points of the best. Methods in red are supervised, whereas InfoShield is unsupervised.

Table 9. InfoShield is Language Independent: Spanish Template from the Twitter Dataset

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

likely will not 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.

6.4 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.

6.4.1 Twitter Data. As shown in Table 9, we find that 23 Spanish tweets are encoded by the given template. The first 22 tweets 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 10, we find that all the tweets are talking about the most popular weekly stories. Although the first half of all 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 parts most worthy of studying.

Table 10. InfoShield Detects Slots: Template from the Twitter Dataset

Constant Slot Insertion Deletion Substitution

T_1	**the**	**mostpopular**	**most popular**	**stories 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 11. Slots Contain User-Specific Information: Template from the HT Dataset

Constant Slot Insertion Deletion Substitution

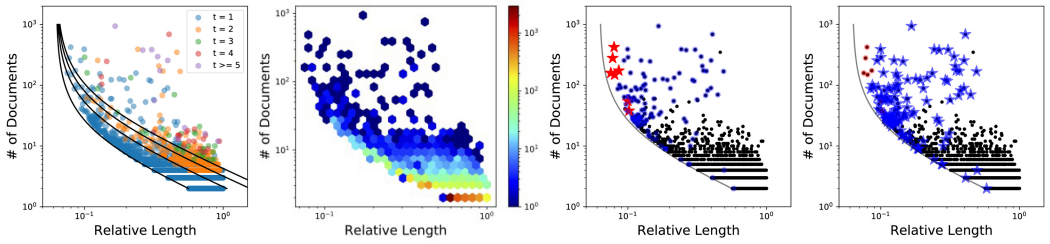
T_1	**not shown for victim's safety**							
#1	(empty)		time		(empty)		(empty)	
#2	personal description		time		(empty)		cost	
#3	(empty)		time		(empty)		cost	
#4	personal description		(empty		preferences		cost	
...18	similar ads							

6.4.2 HT Data. In Table 11, 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—that is, although each slot has a specific purpose in Table 11, the text can be in multiple formats, such as "until 9pm" versus "9 P.M." Work could be done to automatically extract and process the information within each slot, but this is beyond the scope of this work.

6.4.3 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 versus HT clusters? Using the Cluster Trafficking dataset, we illustrate the lower bound of relative length versus the number of documents per cluster in Figure 5(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 5(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



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

Fig. 5. Perpetrators seem separable, thanks to our features. (a) All clusters (circles) and the lower bounds (black lines) are shown—points are above the lower bound, as expected. (b) Heatmap of the same: most points are close to the lower bound. (c) Spam clusters are emphasized as red stars. (d) HT clusters are emphasized as blue stars. Note that the majority of spam and HT clusters (red and blue stars) sit apart from the benign clusters.

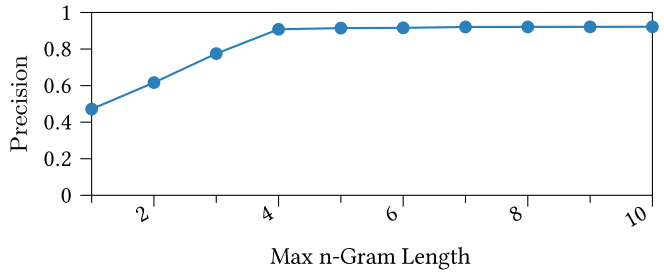


Fig. 6. 5-grams are enough. Precision stabilizes after $n = 4$.

follow patterns in this space. As shown in Figure 5(c), most spam clusters (red stars) have small relative length with a high number of documents; in Figure 5(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) and (2) the outlier clusters that lie far from the lower bounds.

6.5 Q3: Robust

How sensitive is InfoShield-coarse to the length of n-grams we use to calculate tf-idf 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 6.

6.6 Q4: Incremental

How does DeltaShield compare to InfoShield? We run experiments comparing the effectiveness and efficiency of these methods.

6.6.1 DeltaShield-coarse. How do the document-term graphs generated by DeltaShield-coarse compare to the ones generated by InfoShield-coarse? The main difference between these algorithms is the approximation of tf-idf scores in DeltaShield-coarse. To measure the impact of this approximation, we compute the ARI and **Homogeneity (HOM)** scores [48] between the cluster labels produced by DeltaShield-coarse and InfoShield-coarse. A high score signifies that the coarse clusters generated by DeltaShield-coarse are very close to the original coarse

Table 12. DeltaShield-coarse Is Near-Perfect: We Get Almost Exactly the Same Clustering for All Datasets When Processing Ads Incrementally

Twitter Data			**Trafficking Data**		
	Test Set #1	Test Set #2		Trafficking10k	Cluster Trafficking
ARI	99.1	99.9	ARI	97.1	99.2
HOM	99.9	99.9	HOM	96.1	96.5

clusters generated by InfoShield-coarse. We run this experiment for both HT and both Twitter datasets, as shown in Table 12.

We see that all metrics are high, meaning that we do not lose much information by using DeltaShield-coarse and processing ads incrementally.

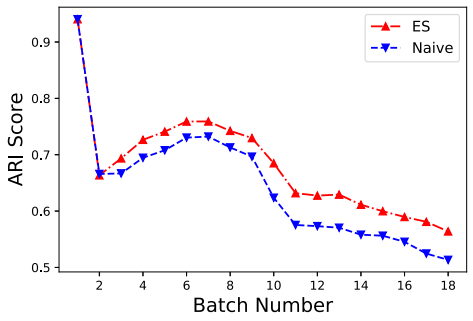
6.6.2 DeltaShield-fine. Here, we compare the effectiveness and efficiency of DeltaShield-fine. We first compare Algorithm 5 (Naive) and Algorithm 6 (ES) to demonstrate that the ES method not only outperforms Naive but also dramatically decreases the running time. We then study the choice of update frequency, which results in a tradeoff between effectiveness and efficiency.

In this experiment, we test an extreme case with the Cluster Trafficking dataset to truly reflect the difference between methods. The dataset is considered as a one big cluster and separated into 18 batches where each batch contains about 2,000 advertisements. Note that InfoShield-coarse is not used in this experiment so that we can stress test DeltaShield-fine with a large number of templates. Since our goal of incremental version is to output as close to the non-incremental one, the ARI score is used as the effectiveness metric here, where the ground truth is clustering labels generated by InfoShield-fine. It is worth noting that this extreme case will not happen if InfoShield-coarse is still implemented, meaning the ARI score is expected to be low. Our empirical result shows that the average number of templates in one cluster generated by InfoShield-coarse is about 3, which is largely smaller than the number in our experiment (as shown in Figure 7(b), it is already more than 200 after batch number 4).

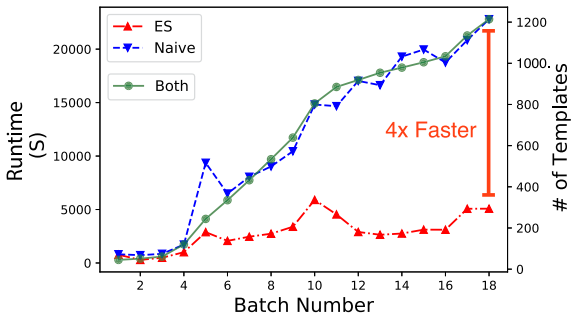
ES versus Naive. The ARI scores over time are shown in Figure 7(a), where we can see that the ARI score of the ES method is always higher than the Naive method after the second batch. As depicted in Figure 7(b), as the number of templates (green line) grows over time, the running time of fitting the templates increases linearly as well. If we compute the slope by the number of templates and running time, the slope of the Naive method is 18, whereas the one of the ES method is 3, which is six times smaller than the Naive method. In Figure 7(c), the ES method achieves a result with only 10% difference comparing to the Naive method, which is more or less a tie. In Figure 7(d), we find that the ES method always outperforms the Naive method more clearly in terms of runtime.

Update Frequency. Next, we study the tradeoff between effectiveness and efficiency. We mainly compare DeltaShield-fine with update frequency every batch and every three batches. In Figure 8(a), we see that as the number of incoming batches increases, the gap between two methods increases as well. Nevertheless, the running time of updating every three batches shown in Figure 8(b) is 1.4 times and 2.8 times faster than the one of updating every batch and InfoShield-fine, respectively. It will substantially mitigate the expensive overhead when the number of clusters and templates are large, which is especially important to the law enforcement where every second counts for them.

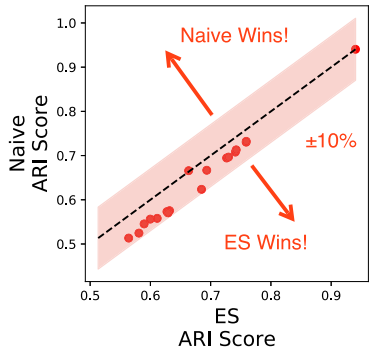
We notice that the low update frequency will slightly hurt the performance. However, we keep in mind that this experiment stress tests DeltaShield-fine, since the number of templates in one



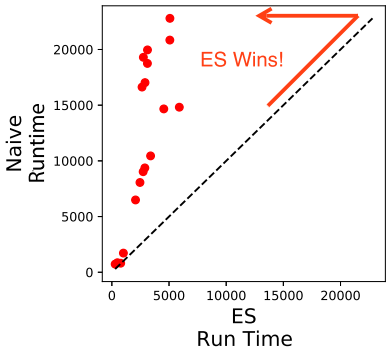
(a) ARI Score over Time



(b) Runtime and Number of Templates over Time



(c) ARI Score: ES vs. Naive



(d) Runtime ES vs. Naive

Fig. 7. DeltaShield-fine Preprocess-ES wins. (a) The ARI score of Preprocess-ES remains higher compared to Preprocess-Naive over time. (b) Demonstration that runtime is highly correlated with the number of templates and shows that Preprocess-ES is much faster. (c, d) Preprocess-ES outperforms Preprocess-Naive in terms of ARI score and runtime, respectively, where each data point denotes the result from one batch.

coarse cluster is much larger than it will be if we first use DeltaShield-coarse. Alternatively, an end user can consider doing the recomputation of all data periodically, depending on their idle time.

7 DISCUSSION AND DISCOVERIES: INFOSHIELD AT WORK

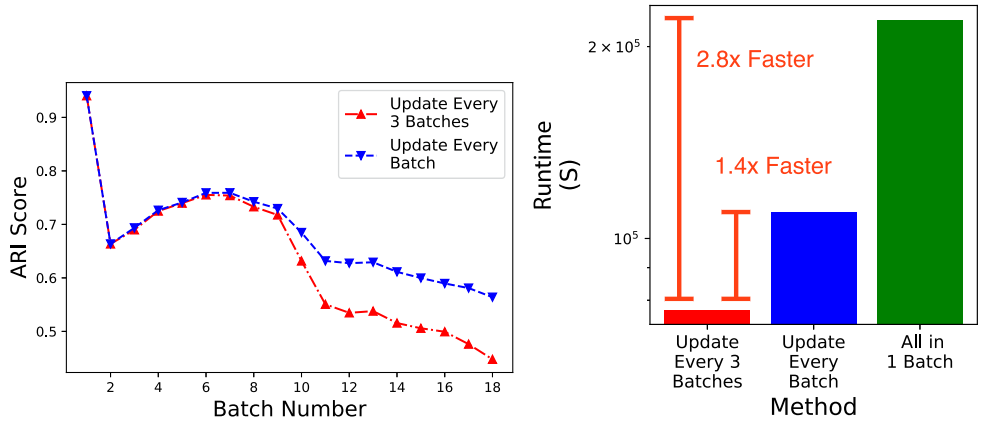
We note that InfoShield has the following advantages.

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 9 is in Spanish, whereas the template in Table 10 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 preclustering and minimizing the total cost. We propose the preceding algorithms because they are scalable and effective on real data.



(a) ARI Score over Time (1 batch = $20K$ ads)

(b) Total Runtime

Fig. 8. DeltaShield-fine offers a strong tradeoff. Even with large update frequency, the accuracy of DeltaShield-fine remains high. (a) Large update frequency loses effectiveness increasingly over time. (b) Increasing the update frequency leads to a much lower runtime.

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.

8 CONCLUSION

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

The main contributions of the method are that it is

- *Practical*, through scalability and using the MDL principle to be parameter-free;
- *Interpretable*, providing a clear visualization and summarization of clusters;
- *Generalizable* and independent of domain (Twitter, HT), as well as of language (English, Spanish etc); and
- *Incremental*, by processing new documents on-the-fly, without having to recompute on historical documents.

In the future, are interested in extending our work on HT detection through spatio-temporal analysis, to understand the movement of possible traffickers through space and time, as well as visualization, so that domain experts can easily interact with the results of our algorithms.

Reproducibility. Code is open sourced at https://github.com/catvajiac/InfoShield-Incremental. The Twitter datasets are public [11]. The Trafficking10k dataset is available after NDA (email Cara Jones at cara@marinusanalytics.com).

REFERENCES

[1] Faraz Ahmed and Muhammad Abulaish. 2013. A generic statistical approach for spam detection in online social networks. *Computer Communications* 36, 10-11 (2013), 1120–1129.

[2] Hamidreza Alvari, Paulo Shakarian, and J. E. Kelly Snyder. 2017. Semi-supervised learning for detecting human trafficking. *Security Informatics* 6, 1 (2017), 1.

[3] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. OPTICS: Ordering points to identify the clustering structure. In *Proceedings of SIGMOD*. 49–60.

[4] Geoffrey J. Barton and Michael J. E. Sternberg. 1987. A strategy for the rapid multiple alignment of protein sequences: Confidence levels from tertiary structure comparisons. *Journal of Molecular Biology* 198, 2 (1987), 327–337.
[5] Regina Barzilay and Lillian Lee. 2003. Learning to paraphrase: An unsupervised approach using multiple-sequence alignment. In *Proceedings of HLT-NAACL*.
[6] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Trans. Assoc. Comput. Linguistics* 5 (2017), 135–146.
[7] Sergey Brin, James Davis, and Héctor García-Molina. 1995. Copy detection mechanisms for digital documents. In *Proceedings of SIGMOD*. 398–409.
[8] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. 1993. Efficient processing of spatial joins using R-trees. In *Proceedings of SIGMOD*. 237–246.
[9] Deepayan Chakrabarti, Spiros Papadimitriou, Dharmendra S. Modha, and Christos Faloutsos. 2004. Fully automatic cross-associations. In *Proceedings of KDD*.
[10] Edwin K. P. Chong and Stanislaw H. Zak. 2004. *An Introduction to Optimization*. John Wiley & Sons.
[11] Stefano Cresci, Roberto Di Pietro, Marinella Petrocchi, Angelo Spognardi, and Maurizio Tesconi. 2017. The paradigm-shift of social spambots: Evidence, theories, and tools for the arms race. In *Proceedings of WWW*. 963–972.
[12] Stefano Cresci, Roberto Di Pietro, Marinella Petrocchi, Angelo Spognardi, and Maurizio Tesconi. 2016. DNA-inspired online behavioral modeling and its application to spambot detection. *IEEE Intell. Syst.* 31, 5 (2016), 58–64.
[13] Clayton Allen Davis, Onur Varol, Emilio Ferrara, Alessandro Flammini, and Filippo Menczer. 2016. BotOrNot: A system to evaluate social bots. In *Proceedings of WWW*. 273–274.
[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv* abs/1810.04805 (2019).
[15] Chris H. Q. Ding and Xiaofeng He. 2004. Principal component analysis and effective *k*-means clustering. In *Proceedings of SIAM DM*. 497–501.
[16] Sara Elmanarelbouanani and Ismail Kassou. 2013. Authorship analysis studies: A survey. *Int. J. Comput. Appl.* 86, 12 (2013), 22–29. https://doi.org/10.5120/15038-3384
[17] Saeideh Shahrokh Esfahani, Michael J. Cafarella, Maziyar Baran Pouyan, Gregory J. DeAngelo, Elena Eneva, and Andy E. Fano. 2019. Context-specific language modeling for human trafficking detection from online advertisements. In *Proceedings of ACL*. 1180–1184.
[18] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of KDD*. 226–231.
[19] Maria Giatsoglou, Despoina Chatzakou, Neil Shah, Alex Beutel, Christos Faloutsos, and Athena Vakali. 2015. ND-sync: Detecting synchronized fraud activities. In *Proceedings of PAKDD*. 201–214.
[20] Nathaniel Gleicher. 2019. How We Respond to Inauthentic Behavior on Our Platforms: Policy Update. Retrieved February 14, 2023 from https://about.fb.com/news/2019/10/inauthentic-behavior-policy-update/.
[21] Peter Grünwald. 2007. *The Minimum Description Length Principle*. MIT Press, Cambridge, NY.
[22] A. Guttman. 1984. R-tree: A dynamic index structure for spatial searching. In *Proceedings of SIGMOD*. 47–57.
[23] Zellig Harris. 1954. Distributional structure. *Word* 10, 23 (1954), 146–162.
[24] Lawrence Hubert and Phipps Arabie. 1985. Comparing partitions. *Journal of Classification* 2, 1 (1985), 193–218.
[25] International Labour Office. 2012. ILO Global Estimate of Forced Labour. Retrieved February 14, 2023 from http://www.ilo.org/wcmsp5/groups/public/---ed_norm/---declaration/documents/publication/wcms_182004.pdf.
[26] Karen Spärck Jones. 2004. A statistical interpretation of term specificity and its application in retrieval. *J. Doc.* 60, 5 (2004), 493–502.
[27] Vani Kanjirangat and Deepa Gupta. 2016. A study on extrinsic text plagiarism detection techniques and tools. *J. Eng. Sci. Technol.* 9, 5 (2016), 150–164. https://doi.org/10.25103/jestr.095.02
[28] Mayank Kejriwal, Jiayuan Ding, Runqi Shao, Anoop Kumar, and Pedro A. Szekely. 2017. FlagIt: A system for minimally supervised human trafficking indicator mining. *CoRR* abs/1712.03086 (2017).
[29] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. 2004. Towards parameter-free data mining. In *Proceedings of KDD*.
[30] Aayushi Kulshrestha. 2021. *Detection of Organized Activity in Online Escort Advertisements*. McGill University (Canada).
[31] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of ICML*. II-1188–II-1196.
[32] Christopher Lee, Catherine Grasso, and Mark F. Sharlow. 2002. Multiple sequence alignment using partial order graphs. *Bioinformatics* 18, 3 (2002), 452–464.
[33] Meng-Chieh Lee, Catalina Vajiac, Aayushi Kulshrestha, Sacha Levy, Namyong Park, Cara Jones, Reihaneh Rabbany, and Christos Faloutsos. 2021. InfoShield: Generalizable information-theoretic human-trafficking detection. In *Proceedings of ICDE*. IEEE, Los Alamitos, CA.

[34] L. Li, O. Simek, A. Lai, M. Daggett, C. K. Dagli, and C. Jones. 2018. Detection and characterization of human trafficking networks using unsupervised scalable text template matching. In *Proceedings of IEEE Big Data*. 3111–3120.
[35] Ming-Ling Lo and Chinya V. Ravishankar. 1994. Spatial joins using seeded trees. In *Proceedings of SIGMOD*. 209–220.
[36] Vitor Martins, D. Fonte, Pedro Rangel Henriques, and Daniela da Cruz. 2014. Plagiarism detection: A tool survey and comparison. *OASIcs* 38 (Jan. 2014), 143–158. https://doi.org/10.4230/OASIcs.SLATE.2014.143
[37] Yasuko Matsubara, Yasushi Sakurai, and Christos Faloutsos. 2014. AutoPlait: Automatic mining of co-evolving time sequences. In *Proceedings of SIGMOD*. 193–204.
[38] Leland McInnes, John Healy, and Steve Astels. 2017. hdbscan: Hierarchical density based clustering. *J. Open Source Softw.* 2, 11 (2017), 205.
[39] M. Mehta, R. Agrawal, and J. Rissanen. 1996. SLIQ: A fast scalable classifier for data mining. In *Proceedings of EDBT*.
[40] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *Proceedings of ICLR*.
[41] Ann Wagner. n.d. Human Trafficking & Online Prostitution Advertising. Retrieved February 14, 2023 from XXX.
[42] Thorn. 2015. A Report on the Use of Technology to Recruit, Groom and Sell Domestic Minor Sex Trafficking Victims. Retrieved February 14, 2023 from http://www.thorn.org/wp-content/uploads/2015/02/Survivor_Survey_r5.pdf.
[43] Chirag Nagpal, Kyle Miller, Benedikt Boecking, and Artur Dubrawski. 2017. An entity resolution approach to isolate instances of human trafficking online. In *Proceedings of NUT@EMNLP*. 77–84.
[44] Rebecca S. Portnoff, Danny Yuxing Huang, Periwinkle Doerfler, Sadia Afroz, and Damon McCoy. 2017. Backpage and bitcoin: Uncovering human traffickers. In *Proceedings of KDD*.
[45] Reihaneh Rabbany, David Bayani, and Artur Dubrawski. 2018. Active search of connections for case building and combating human trafficking. In *Proceedings of KDD*.
[46] J. Rissanen. 1978. Modeling by shortest data description. *Automatica* 14, 5 (Sept. 1978), 465–471.
[47] Jorma Rissanen. 1983. A universal prior for integers and estimation by minimum description length. *Ann. Stat.* 11, 2 (1983), 416–431.
[48] Andrew Rosenberg and Julia Hirschberg. 2007. V-Measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of EMNLP-CoNLL*. 410–420. https://www.aclweb.org/anthology/D07-1043.
[49] Neil Shah, Hemank Lamba, Alex Beutel, and Christos Faloutsos. 2017. The many faces of link fraud. In *Proceedings of ICDM*. IEEE, Los Alamitos, CA, 1069–1074.
[50] Karishma Sharma, Feng Qian, He Jiang, Natali Ruchansky, Ming Zhang, and Yan Liu. 2019. Combating fake news: A survey on identification and mitigation techniques. *ACM Trans. Intell. Syst. Technol.* 10, 3 (2019), 1–42.
[51] Siwei Shen, Dragomir R. Radev, Agam Patel, and Güneş Erkan. 2006. Adding syntax to dynamic programming for aligning comparable texts for the generation of paraphrases. In *Proceedings of COLING/ACL*. 747–754.
[52] Edmund Tong, Amir Zadeh, Cara Jones, and Louis-Philippe Morency. 2017. Combating human trafficking with multimodal deep models. In *Proceedings of ACL*. 1547–1556.
[53] Chao Yang, Robert Chandler Harkreader, and Guofei Gu. 2013. Empirical evaluation and new design for fighting evolving Twitter spammers. *IEEE Trans. Inf. Forensics Secur.* 8, 8 (2013), 1280–1293.
[54] Xinyi Zhou and Reza Zafarani. 2020. A survey of fake news: Fundamental theories, detection methods, and opportunities. *ACM Comput. Surv.* 53, 5 (2020), 1–40.

Received 3 September 2021; revised 19 February 2022; accepted 8 May 2022