Tracking Cyber Threat Actors in Semi-Automatic OSINT Analysis

Hanna Lilja **Lukas Lundmark**

Swedish Defence Research Agency (FOI)
SE-164 90 Stockholm
SWEDEN

hanna.lilja@foi.se lukas.lundmark@foi.se

ABSTRACT

The increasing amount of information shared online, not least on social media, in the form of unstructured text data provides an opportunity to complement traditional sources of cyber threat intelligence. As such vast volumes of data cannot be processed manually, we explore some possibilities of using machine learning to assist in the analysis. We specifically focus on the retrieval of information related to named threat actors. By fine-tuning existing language models to specific downstream tasks, based on pseudo-automatically annotated data, models for detection and extraction of previously unseen threat actors are obtained. We perform multiple evaluations under varying conditions, some of which suggest the models are indeed capable of producing results that could be useful in a setting of semi-automatic analysis. In addition, we view this a case study of the application of general language models to a niche and domain-specific task, and reflect on some more general lessons learned.

1.0 INTRODUCTION

Threat intelligence is an integral part of cyber defence. Logging and monitoring of technical systems are traditional sources of cyber threat intelligence (CTI). The increasing amount of information shared online, not least on social media, provides an opportunity to complement traditional sources to improve situational awareness in the cyber environment. Leveraging these new sources at scale requires an ability to sift through vast amounts of unstructured data at a much higher rate than any analyst could manage. Hence, there is a need for semi-automatic analysis, where the strengths of the analyst's mind are combined with the processing power of computers.

Recent developments in natural language processing (NLP) and machine learning (ML) have provided powerful and versatile language models which represent a general understanding of language, obtained through massive amounts of text data and computing power. These models can be fine-tuned on much smaller amounts of data in order to learn a specific task. In this work we explore the possibilities of applying such language models to the CTI context, in particular we focus on the task of automatically identifying (previously unseen) cyber threat actors mentioned in text. In addition to the CTI-related benefits of such a capability, this also serves as a case study of the application of general language models to a niche and domain-specific task, for which pre-existing data sets and evaluation benchmarks cannot be assumed to be available.

2.0 BACKGROUND

Previous work on automatic extraction of CTI from large amounts of OSINT text data include a system presented in [1]. This system is capable of generating warnings regarding certain types of cyber threats based on data from Twitter and darkweb forums, using a term- and dictionary-based text analysis approach. The effectiveness of the method is demonstrated in two scenarios, related to vulnerabilities and data breaches respectively.

In this work we use pre-trained language models. In NLP, language models are machine learning models that learn a probabilistic representation of language. These models train in an unsupervised fashion, where the model is given a corpus of text and an unsupervised pretext task. Pretext tasks are tasks that do not require any human annotation. A common example is the Language Model Objective (LMO), where the model is tasked with predicting the next word in a given sequence of text. The purpose of pretext tasks is not to solve the task itself, but to prepare the model for fine-tuning on a downstream task.

Most recent language models are based on a form of neural network called the Transformer [7]. Unlike previous methods, which process input-sequences one element at a time, the Transformer can process all elements in the sequence in parallel. This more efficient form of text processing has made it possible to use more training data and larger models. This, in turn, has contributed to Transformer-based language models achieving state-of-the-art performance in various NLP tasks.

In this work we evaluate four different Transformer-based language models.: *Bidirectional Encoder Representation from Transformer (BERT)* [2] is the approach that popularized pre-trained transformers. BERT uses two pretext tasks, *Masked Language Model* (MLM) objective and *Next Sentence Prediction* (NSP). For MLM, a subset of tokens in the input texts are masked, and the model must predict the original tokens. In NSP, the model predicts if two text sequences follow each other in the training corpus. All models used in this paper, except for DistilBERT, have pre-training setups using two pretext tasks. BERT is available in two sizes: BERT-base model with 110 million parameters and BERT-large with 340 million parameters. Due to hardware constraints during both training and evaluation, we use the smaller version, BERT-base. This model is pre-trained utilizing a combination of BookCorpus[1] and English Wikipedia.

RoBERTa [4] is an improved version of BERT with minor adjustments to the text representation, training schema, and the training corpora. While the original BERT model was trained on English Wikipedia and BookCorpus, RoBERTa was, in addition, trained on about 160GB of news articles and web text. RoBERTa also differs in how it represents text, using a Byte Pair Encoder (BPE) tokenizer [6]. The BPE is better at handling out-of-vocabulary words than BERT's WordPiece tokenizer, thus making it suitable for online texts with diverse vocabularies. Similar to BERT, we use the smaller version with 110 million parameters.

DistilBERT [8] is a distilled version of BERT. Distillation, in the context of machine learning, is the process of training smaller models (students) with the output of larger models (teachers) as input. Distillation can help reduce the size of the model while still retaining its performance. The smaller size of the models can prevent overfitting on downstream tasks. DistilBERT is trained using the output of BERT-base and contains 40% fewer parameters while still retaining circa 97% of BERT-base's performance.

BERTweet [5] is a modified version of RoBERTa pre-trained on 850 million English language tweets. BERTweet also has a unique tokenizer and pre-processing scheme to handle commonly occurring elements of tweets, such as hashtags, emojis, and usernames.

3.0 METHOD

Before a language model can be fine-tuned, a learnable task must be defined, and data reflecting the concepts of that task obtained. We use the STIX (Structured Threat Information Expression)[2] data format as a model of the CTI context, and as starting point for identifying types of information which would be of interest to find. Using an existing and established framework not only saves time and effort, but also helps to facilitate compatibility between information obtained through structured and unstructured data. STIX consists of a number of domain objects (e.g., *Threat Actor*, *Attack Pattern*, *Malware*), along with attributes, and relationships between objects. Two of these domain objects are of particular interest here, *Threat Actor* and

[1] S. Kobayashi, *Homemade BookCorpus*, 2018.

[2] *Introduction to STIX*, https://oasis-open.github.io/cti-documentation/stix/intro.html.

Intrusion Set. The former represents the actors (e.g., individual, group, organization) behind the threat, whereas the latter is a more abstract representation of the threat itself. They both, however, relate to named entities. While in theory the difference is well defined, that does not guarantee that it is reflected well in data collected in-the-wild. Based on this observation it was decided that the object of study, from now on *threat actor*, should cover the concepts of both *STIX Threat Actor* and *STIX Intrusion Set*.[3] With the task defined the relevant data set can be crafted (Section 3.1) and machine learning models trained (Section 3.2).

3.1 Crafting the Data Set

The raw data is collected through the Twitter steaming API, based on a set of generic cyber-related keywords such as "cyber" and "malware". This generates a large and highly imbalanced set of data, in which only a small fraction of the instances are related to threat actors. Supervised machine learning requires labelled data, in this case positive examples of tweets mentioning threat actors, and negative examples with no such mentions. Manual annotation of data is typically very time consuming, and therefore a potential blocker in the process of applying ML to specific tasks. We instead opt to explore the possibilities of using rule-based NLP techniques to annotate data in a pseudo-automatic[4] fashion (Section 3.1.1), combined with a manually supervised data cleaning effort (Section 3.1.2). We use only the text content of the tweets, and no meta-data.

3.1.1 Pseudo-Automatic Annotation

Rule-based NLP techniques depend on manually defined language patterns and are therefore relatively simple and limited. As opposed to data-driven techniques, however, they have the benefit of not requiring any data upfront. For this reason, they are valuable as a quick way of starting data collection and exploring new data. With regards to threat actors, rule-based NLP is an efficient tool for finding mentions of actors whose names are already known. We are thus able to automatically sort downloaded tweets into two categories, texts which contain the name of a known threat actor, and texts which do not. While this annotation method undoubtedly saves time and effort, it also increases the risk of inaccurate labels. Any tweets containing (only) unknown threat actors, i.e., names that are not "on the list", will become false negative examples. A false positive example is created when the name of a known threat actor also has other meanings and is used in a different context.

3.1.2 Data Cleaning

The initial pseudo-automated annotation has a high rate of false positives. For example, texts like "Don't forget our Platinum Cyber Mondays Special Sale" are given positive labels due to the words *cyber* (a Twitter stream keyword) and *Platinum* (also the name of a threat actor). To reduce the impact of this problem, we first manually inspect a subset of the positive tweets. Based on these false examples, we construct additional word-based filtering rules to remove reoccurring errors.

Furthermore, a large bulk of cyber tweets are produced by automated Twitter accounts which retweet cyber-related news articles. As a result, our initial data set contains multiple tweets referencing the same original article but generated by different accounts. Such tweets are often identical – differing only in how much of the original article they cite, the hashtags they use, and the format of any URLs. The size of these groups of near-duplicate tweets, hereafter referred to as *duplicate groups*, can range from 2 to 15. These duplicate groups can cause a wide range of problems during both training and evaluation. They can take up a large fraction of all samples for less common threat actors, thus likely overfit the model to those tweets. Similarly, if the evaluation set contains many duplicates, it can skew the evaluation results.

[3] While this to some extent invalidates the information compatibility argument made earlier, the ambiguity here is at least well defined, and could for example be resolved by an analyst in a setting of semi-automatic analysis.

[4] We use the term *pseudo-automatic* to emphasise the fact that the resulting set of labels is an approximation of the truth, and that the process is paired with a step of manually supervised data cleaning to reduce the number of incorrect labels.

We utilize MinHash [10], a Locality Sensitive Hashing algorithm that finds similar documents in close-to-linear time to remove these duplicates. We use the implementation found in the *datasketch* python library[5]. Since we are looking for near-perfect copies, we use a large shingle size of k=9. If two tweets have shingle sets with a Jaccard similarity [3] over 0.4, we regard them as duplicates. To find all duplicates of a tweet, we conceptualize the tweets in our data set as a graph, where tweets are nodes, and two tweets are connected if they are duplicates. We then find all *connected components* by traversing the graph using Depth First Search, and keep only the longest tweets in each connected component, discarding the remaining tweets.

Finally, we clean up the contents of each tweet. Some distracting elements – hyperlinks, usernames, and e-mail addresses – are replaced with corresponding masking tokens since they do not contain task relevant information. Similarly, emojis are transformed to text descriptions, e.g., 👍 is converted to :thumbs_up. This allows models that do not have emojis in their vocabularies to understand the sentiment of the character.

3.2 Fine-Tuning Language Models

The task of finding named threat actors mentioned in text is divided into two subtasks. The first, which we call *threat actor detection*, is a binary classification task. The objective is for the model to determine if a given text (tweet) mentions a threat actor or not. The goal of the second task, which we call *threat actor extraction*, is to identify the name of the threat actor.

3.2.2 Data Splits

The purpose of using machine learning in the threat actor context is to obtain the ability to automatically find *new threat actors*[6] when mentioned in text. We therefore want to avoid a situation where the models just memorizes the actual names of the threat actors in the training data. We also want to ensure that evaluation reflects the models' ability to generalize to previously unseen threat actor names. To this end we construct the training, validation, and test splits of our data so that no threat actor name occurs in more than one split. The threat actors in our data set have a power-law-like distribution – a small minority of threat actors make up the bulk of positive examples. To create the splits, we sort the threat actors based on frequency of occurrence in the data set. For the training split, we select the most frequent threat actors and their corresponding tweets until we have acquired at least 80% of all positive tweets. From the remaining threat actors we keep selecting the most frequent until we have reached at least 10% of all positive tweets for the validation split. The remaining threat actors are then used for the test split. Any tweet containing actors from two or more splits is then removed to prevent overlap. The resulting training split included 16 different actors, the validation split 6, and the test split 48. A total of around 3300 positive tweets were used (reduced from over 8000 through the cleaning process).

3.2.3 Continued Pre-Training

Except for BERTweet, our pre-trained language models are pre-trained on corpora which share little-to-no similarity with our target domain of cyber tweets. As a result, these models are likely to experience a domain shift when fine-tuned on our data, causing them to underperform. One way to tackle this is continued pre-training [11]. The idea is to continue unsupervised pre-training on a new data set sampled from the target domain. This continued pre-training adjusts the model to the target domain, thus diminishing the domain shift experienced by the model during subsequent fine-tuning.

For this reason, additional data for continued pre-training was sampled from the downloaded cyber tweets. Again the previously described data cleaning method was used, and tweets that overlapped with the

[5] E. Zhu och V. Markovtsev, *ekzhu/datasketch: First stable release,* Zenodo, 2017.

[6] All names of threat actors that are known to us can easily be listed and thereafter found in text through the use of rule-based NLP techniques, we do not need machine learning for that.

annotated data set were filtered out. To investigate how the size of the pre-training set affects downstream performance, we compiled two data sets of different sizes – one with 25,000 tweets and one with 250,000 tweets. We then generated three different instances for each of the four language models, one with no additional pre-training and one for each pre-training data set. The *Masked Language Model*-objective was used as pre-training objective.

3.2.4 Threat Actor Detection

The task of threat actor detection entails classifying whether a tweet contains one (or more) mentions of a threat actor or not. We train each language model with the data splits described in Section 2.2.2. Negative examples for each split were sampled from the set of downloaded cyber tweets. To prevent overfitting to a small set of negative samples, we used a ratio of 4-to-1 negative-to-positive samples in the training set.[7] A larger fraction of negative training than that was observed to produce very low recall, while a much smaller fraction increased the risk of the model overfitting. For the validation and test set the distribution between positive and negative examples was kept even.

3.2.5 Threat Actor Extraction

Threat actor extraction is a token-level[8] classification task in which each token is either part of a threat actor name or not. For token-level annotation, the tweets are tokenized and the tokens overlapping the known threat actor names are given a positive label. Training is similar to the detection task, except for an additional dropout layer between the token output and the final classifier layer. Due to time constraints, for this task we use only the RoBERTa language model since its tokenizer is the easiest one to extract correct labeling from. To reduce the task's difficulty, we assume that each tweet the model receives during inference contains at least one threat actor, and therefore only include such tweets during training. We consider this simplification justified since the threat actor detection model is meant to solve the task of filtering out tweets that mention threat actors.[9]

3.2.6 Training and Hyperparameter Tuning

As is standard when training Transformers [7], we start with a warmup phase – the learning rate is initialized to close to 0 and increases until it reaches the maximum learning rate after *n* steps. We also use linear learning rate decay, where the learning rate decreases after each step until it reaches 0. All model instances are fine-tuned for a single epoch. We perform hyperparameter tuning over four parameters – weight decay factor, number of warmup-steps, maximum gradient norm, and maximum learning rate. For the extraction task we also search for the optimal dropout rate for the final classification layer. We use Bayesian Optimization to explore the parameter space[10]. For each model, we perform 50 iterations and select the model which achieves the highest F1-score on the validation set.

[7] It is worth noting that this still constitutes a significant overrepresentation of positive examples, compared to the natural distribution.

[8] Before text is "read" by a language model, the input string is split into continuous segments known as tokens, often consisting of words or sub-words. Different models have different ways of doing this, e.g., splitting on white space or splitting based on a pre-defined vocabulary. Rare words, such as threat actor names, tend to be divided into mutliple tokens. This is why threat actor extraction is performed on token-level, rather than word- or character-level.

[9] When we later deploy the models, we first let the detection model find positive samples in a stream of tweets, which are then analysed by the extraction model.

[10] We used the hyperparameter optimization library Optuna [9].

4.0 RESULTS

Two different sets of data are used to evaluate the performance of the models. The first is a held-out partition of the pseudo-automatically annotated data that was used for training, a common practice in the field of machine learning. This evaluation is described in Section 4.1. The second is a set of data collected at later point in time, and therefore not overlapping with the first one. For the second set of data the evaluation is done with regards to manually applied labels, described in Section 4.2.

4.1 Evaluation on Pseudo-Automatically Annotated Data

We evaluate performance on the threat actor detection task for each language model (and pre-training condition) on our held-out test set. The results are presented in Table 4-1, the metrics used are precision, recall, and F1-score (which is the harmonic mean of precision and recall). We note that all models are roughly similar in performance across each metric. The models pre-trained on tweets and web text (BERTweet and RoBERTa) seem to have a slight advantage, particularly in terms of recall. The effect of additional pre-training ranges from negligible to slightly positive. The best model with regards to F1-value is the RoBERTa model pre-trained on 250K tweets. However, BERTweet without any additional pre-training has the highest precision, while still achieving a reasonable recall.

Table 4-1: Evaluation results for the *threat actor detection* task on the pseudo-automatically annotated data.

Language Model	*# Pre-training Tweets*	*Precision [%]*	*Recall [%]*	*F1 [%]*
BERT	0	95.5	88.1	91.7
	25k	91.5	89.4	90.4
	250k	94.5	87.6	90.9
DistilBERT	0	94.8	83.9	89.0
	25k	92.3	88.1	90.2
	250k	92.9	90.0	91.4
RoBERTa	0	93.6	93.6	93.6
	25k	95.5	88.1	91.7
	250k	93.7	95.0	94.3
BERTweet	0	96.0	91.3	93.6
	25k	95.2	92.6	93.9
	250k	95.2	92.2	93.7

The results for the threat actor extraction task (precision, recall, and F1-score on token-level) are presented in Table 4-2. Here we notice a more apparent positive effect of additional pre-training, a five percentage point increase in F1-score between no and full pre-training. We observe two possible reasons for the low recall. First, the model has difficulty with threat actor boundaries, e.g., the last few tokens in long threat actor names are often ignored. Second, the model often misses threat actor mentions in hashtags, which are much harder to detect since there is little surrounding language context to help the model.

Table 4-2: Evaluation results for the *threat actor extraction* task on the pseudo-automatically annotated data.

Language Model	*# Pre-training Tweets*	*Precision [%]*	*Recall [%]*	*F1 [%]*
RoBERTa	0	71.1	15.6	25.6
	25k	67.7	16.7	26.8
	250k	72.1	19.5	30.7

4.2 Evaluation through Manual Annotation

As a complement to the evaluation on the pseudo-automatically annotated set of data, additional evaluation of the RoBERTa-based model (with extra pre-training on 250k cyber-related tweets) is performed through manual annotation, where manually applied labels are compared to the results of the detection and extraction models. A set of 800 newly downloaded tweets was annotated, each tweet being read by three individuals[11] who would classify it as positive or negative and (in the positive case) record the name of the threat actor. A majority vote decided the final classification label. The only data cleaning performed on this data was manual removal of duplicates.

The evaluation results for the detection task are shown in Table 4-3, where threshold value indicates how certain (on a scale from 0 to 1) the model must be that the analysed text does indeed mention a threat actor for it to be regarded as a positive example (in the evaluation on the pseudo-automatically annotated data the threshold value 0.5 is used throughout). The metrics used are once again precision, recall, and F1-score. The data which was manually annotated was sampled to be balanced between positive and negative predictions from the detection model at threshold value 0.5. This distribution is significantly different from the natural one, in our stream of cyber-related tweets only a very small fraction contains threat actors.

Table 4-3: Evaluation results through manual annotation for the *threat actor detection* task at different threshold values.

Language model	*Threshold*	*Precision [%]*	*Recall [%]*	*F1 [%]*
RoBERTa 250k	0.99	19.1	80.4	30.8
	0.9	15.9	91.1	27.1
	0.75	15.1	98.2	26.2
	0.5	14.0	100	24.6

The model for threat actor extraction returns a value for each token in the text, indicating the probability for that token to be part of a threat actor name. This token-level information can be used to produce name suggestions consisting of complete words from the original text. These name suggestions, *(threat actor) candidates*, can be obtained through more or less strict conditions on the machine learning results. Here we test four such sets of conditions, all of which take into account both the how certain it is that the text contains a threat actor (detection threshold value) and the probability for individual tokens to be part of the name of a threat actor. The candidate types $cand_H$, $cand_M$, and $cand_L$ use the detection threshold values 0.9, 0.75, and 0.5 respectively. The candidate type $cand_F$ also uses the threshold value 0.5, but is forced to find at least one candidate in each text. The conditions on token-level probability also differs between candidate types.

The results of comparing[12] the obtained candidates with the manually recorded threat actor names are presented in Table 4-4. The precision, recall, and F1-score metrics are calculated over the whole set of manually annotated tweets. An additional metric, which we call *I-precision*, is also introduced. This is the precision taking only those tweets into account which the annotators jointly considered to be mentioning a threat actor. This does not reflect the performance of the extraction model in reality, but gives us an idea of how it might perform under ideal conditions, i.e. in tandem with a perfect detection model[13].

[11] These individuals are not cyber security experts. Given the fact that tweets generally tend to be short and not necessarily convey a clear statement, this is a relatively challenging annotation task.

[12] This comparison was done in a somewhat more forgiving manner than that done for the detection task. A candidate was seen as correct if it (in all essential parts) overlapped a name suggestion given by at least one annotator.

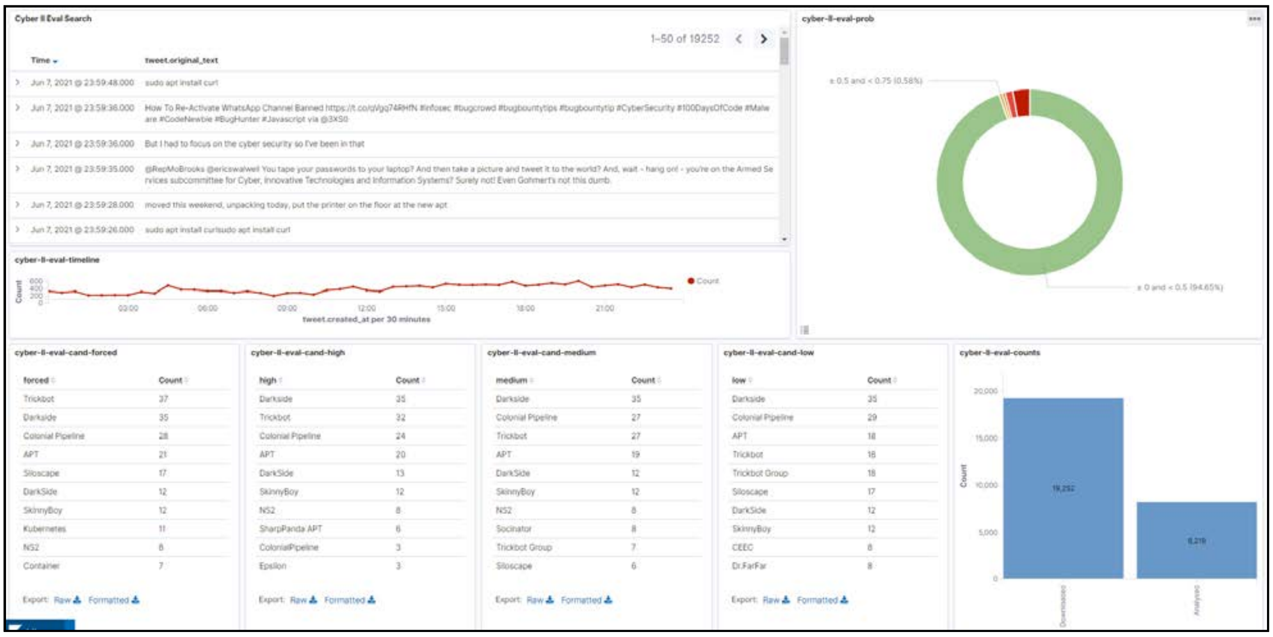
[13] The extraction model, which was trained on positive examples only, to some extent assumes that the detection task is solved earlier on in the analysis flow. An underperforming detection model can therefore have a negative impact on the quality of the extraction results.

Table 4-4: Evaluation results through manual annotation for the *threat actor extraction* task.

Candidate type	*Precision [%]*	*Recall [%]*	*F1 [%]*	*I-precision [%]*
$cand_H$	11.4	61.5	19.3	53.3
$cand_M$	10.1	69.2	17.6	53.6
$cand_L$	8.4	76.9	15.1	51.5
$cand_F$	8.7	73.8	15.6	53.9

The results for the detection task differ significantly from those based on the pseudo-automatically annotated data (comparable at threshold value 0.5). In particular there is a massive drop in precision, whereas the recall is slightly increased. The results for the extraction task are not directly comparable, but the trend of low precision and higher recall is the same. The fact that the I-precision values are significantly higher indicates that the low precision of the detection model is a major contributing factor to the very low precision of the extraction model.

To add one more perspective to the performance of the models, we conduct one additional evaluation, meant to reflect more user-like conditions, with the help of a purpose-built dashboard (see Figure 4-1).



Cyber II Eval Search

1–50 of 19252

Time	tweet.original_text
Jun 7, 2021 @ 23:59:48.000	sudo apt install curl
Jun 7, 2021 @ 23:59:36.000	How To Re-Activate WhatsApp Channel Banned https://t.co/qVgq74RHfN #infosec #bugcrowd #bugbountytips #bugbountytip #CyberSecurity #100DaysOfCode #Malware #CodeNewbie #BugHunter #Javascript via @3XS0
Jun 7, 2021 @ 23:59:36.000	But I had to focus on the cyber security so I've been in that
Jun 7, 2021 @ 23:59:35.000	@RepMoBrooks @ericswalwell You tape your passwords to your laptop? And then take a picture and tweet it to the world? And, wait - hang on! - you're on the Armed Services subcommittee for Cyber, Innovative Technologies and Information Systems? Surely not! Even Gohmert's not this dumb.
Jun 7, 2021 @ 23:59:28.000	moved this weekend, unpacking today, put the printer on the floor at the new apt
Jun 7, 2021 @ 23:59:26.000	sudo apt install curlsudo apt install curl

cyber-II-eval-cand-forced

forced	Count
Trickbot	37
Darkside	35
Colonial Pipeline	28
APT	21
Siloscape	17
DarkSide	12
SkinnyBoy	12
Kubernetes	11
NS2	8
Container	7

cyber-II-eval-cand-high

high	Count
Darkside	35
Trickbot	32
Colonial Pipeline	24
APT	20
DarkSide	13
SkinnyBoy	12
NS2	8
SharpPanda APT	6
ColonialPipeline	3
Epsilon	3

cyber-II-eval-cand-medium

medium	Count
Darkside	35
Colonial Pipeline	27
Trickbot	27
APT	19
DarkSide	12
SkinnyBoy	12
NS2	8
Socinator	8
Trickbot Group	7
Siloscape	6

cyber-II-eval-cand-low

low	Count
Darkside	35
Colonial Pipeline	29
APT	18
Trickbot	18
Trickbot Group	18
Siloscape	17
DarkSide	12
SkinnyBoy	12
CEEC	8
Dr.FarFar	8

Figure 4-1: A dashboard designed for manual evaluation of machine learning models. It shows, among other things, the percentage of tweets classified as containing threat actors at different threshold values (pie chart), and top-10 lists for threat actor candidates, for a selected time interval.

A fresh batch of Twitter data was downloaded and analysed by the machine learning models, over the course of two weeks. The results were then studied in 14 data intervals of 24 hours. For each interval the proportion of tweets classified as positive was recorded. The top-10 most frequent threat actor candidates for each candidate type were then manually[14] sorted into the following categories: *threat actor*, *malware*, *organization*, *person*, *acronym*, *other*. The categories were chosen to cover the types of named entities we by experience knew were common among miss-classifications. The result of the manual categorization of top-10 candidates is illustrated in Figure 4-2, which for each candidate type and category shows the distribution of number hits per time interval (i.e., 14 data points for each category).

[14] This work was done by one person only, however, as the dashboard allows access to multiple tweets for each candidate and thus provide more context it can be argued that this annotation is less subjective in nature than the one previously described.

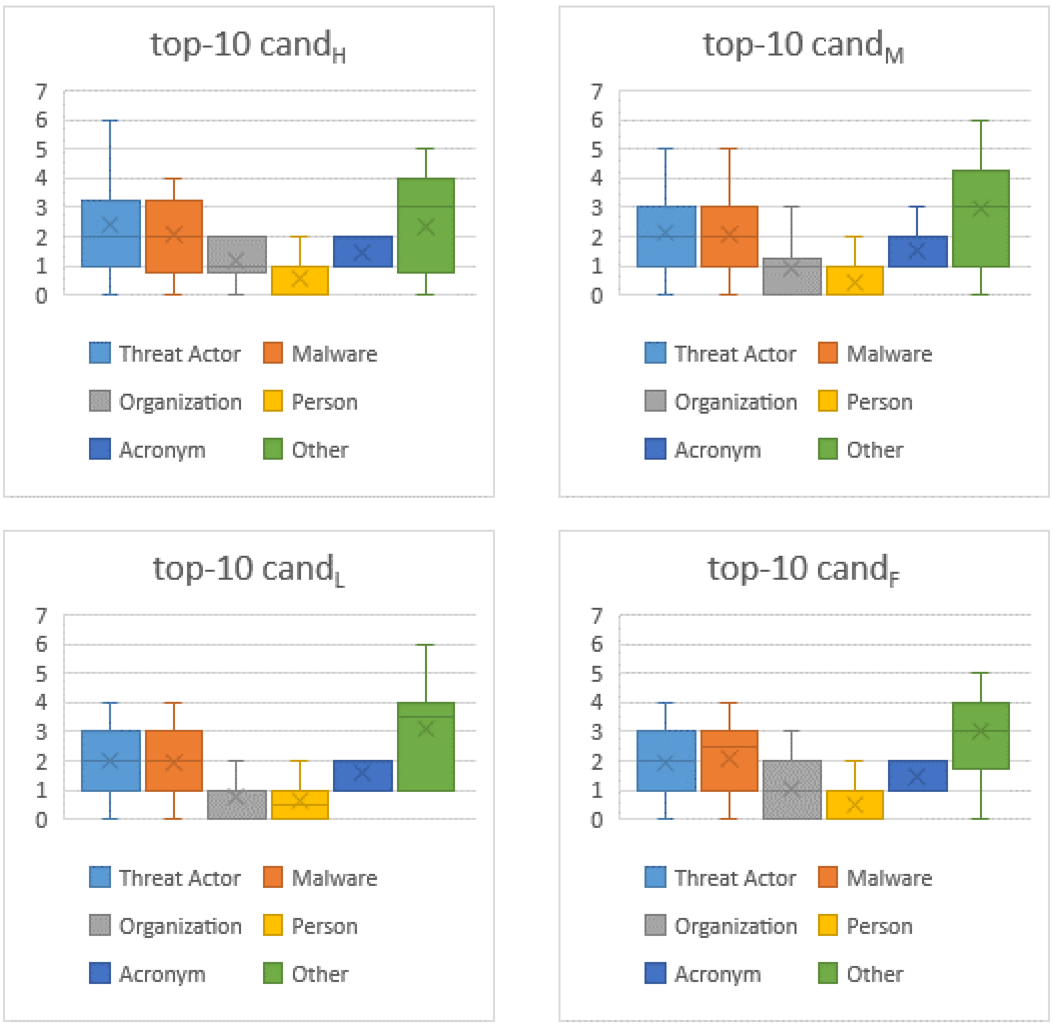


Figure 4-2: Distribution of number of hits per 24-hour interval for each entity category, in the top-10 lists for each threat actor candidate type (*$cand_H$*, *$cand_M$*, *$cand_L$*, *$cand_F$*).

The proportion of tweets classified as positive at threshold value 0.5 varies between 4.0% and 8.4%, with a mean of 5.4% over the whole time period. In other words, close to 95% of the tweets are filtered out, even at a low threshold. Since the natural distribution is very imbalanced this is good sign, it means that the model is significantly closer to the real distribution between positive and negative than the training data was. This is also encouraging with regards to the earlier indications of high recall[15], as they are not obviously explained away by a massive overrepresentation of positive classifications.

For *$cand_H$*, which has the strictest probability requirements, we see the highest mean number of actual threat actors among the top-10 at 2.4 (the other candidate types all have a mean around 2). The median for number of threat actors is 2 for all candidate types. All types has 0 threat actors in at least 1 of the 14 intervals, *$cand_H$* has at most 6 within one interval, *$cand_M$* at most 5, and the other two at most 4. Regarding confusion with other entity categories the pattern is the same for all candidate types. The most common specific error the model makes is to mistake the name of a *malware* for the name of a *threat actor*. This is not surprising, since the language context of these entity types can be very similar or even identical – even a human reader may need background knowledge to distinguish between them. The *other* category is the most prevalent one, but no additional error categories were identified.

[15] True recall is in reality impossible to calculate with huge data sources like Twitter, since we are unable to measure it over the entire set of data.

5.0 DISCUSSION

The evaluation on the held-out test set yields very promising results in terms of both precision and recall, while the results of the second evaluation indicate that the precision is severely lacking. Additional iterations of data collection, annotation, and training would likely improve the performance. The reasonably high recall suggests that the detection model is capable of performing a relatively efficient filtering of tweets, where a large proportion of the interesting ones (in this case, those that mention threat actors) are kept while a large amount of the irrelevant ones can be automatically discarded. Most of the threat actor candidates provided by the extraction model are inaccurate, regardless of candidate type, but a top-10 list typically include one or two relevant suggestions. In a setting of semi-automatic analysis this could possibly be useful to some extent. One possible way of improving performance in the extraction task in particular, would be to leverage existing models for general named entity recognition (NER), either as a starting point for fine-tuning or in an ensemble arrangement alongside our model.

The results of the different evaluations offer some insights regarding the possibilities and challenges of applying machine learning and language models to a domain-specific and niche problem for which there are no available data sets or established evaluation benchmarks.

The large difference between the results of the two evaluations indicates that the underlying data sets represent different sections of the complete data stream (which is hardly surprising considering the fact that they were constructed in different ways and at different points in time) and that the models have not learned to generalize to a large enough extent. This underlines the importance of thinking through how data sets are sampled, especially when the distribution of data must deviate significantly from the natural one.

Rule-based NLP techniques are useful tools for pseudo-automatic annotation of data, which is needed to enable data driven methods like supervised machine learning. If a noisy data source (such as Twitter) is used, some manual data cleaning may be necessary to make the data useable. It is important to consider not just what is included in the data set, but also if something is missing. In the threat actor case one potential path to improvement we have identified would be to include “better” negative examples in the training data, for example by specifically collecting tweets mentioning named malware, in an effort to help the model better distinguish between different types of entities.

Another important question is whether the machine learning problem should be split into subtasks, and if so, how. In this case we use two models, one for detection and one for extraction of threat actors. It would be possible to split the task even further, for example by adding a model trained to determine if a tweet is at all related to cyber security to the beginning of the chain. Such a model could potentially filter out irrelevant topics, like “cyber bullying”, and thereby making the task of the threat actor detection model more clear-cut (as it can focus more on what differentiates threat actors from other cyber security-related concepts). However, relying too much on models earlier on in a chain comes with the risk of early errors propagating and confusing models further down the line. Our extraction model shows symptoms of this, as its performance (unsurprisingly) appears to be negatively affected by the poor precision of the detection model. Another option would, in this case, be to let the extraction model see large amounts of negative examples at training time and so hopefully teaching it to better deal with future detection model errors. Generally speaking, some experimentation may be needed for a new application when it comes to task design. Also, iteratively annotating data and retraining model as subtask performance is improved may be beneficial.

How model performance should be evaluated is not always obvious, different evaluations may yield different results, as shown in the threat actor case. In the absence of established benchmarks, which are unlikely to exits for a niche problem, one must set the framework for the evaluation oneself. It is important to consider what potential biases and sources of errors a particular evaluation scheme might introduce. It may also be beneficial to not only look at classic metrics, but to also reflect on how the results may be used and what that means in terms of performance requirements.

6.0 CONCLUSIONS

The use of pseudo-automatic annotation of data, through rule-based NLP techniques, for fine-tuning of language models in the context of cyber threat actor detection and extraction appears to be viable. However, the results of the secondary evaluation indicate that multiple iterations of data collection, annotation, and training may be needed. Evaluation under more user-like conditions suggests that the models are capable of producing results that could potentially be useful in a setting of semi-automatic analysis.

7.0 REFERENCES

[1] A. Sapienza, A. Bessi, S. Damodaran, P. Shakarian, K. Lerman, and E. Ferrara," Early Warnings of Cyber Threats in Online Discussions," in *2017 IEEE International Conference on Data Mining Workshops (ICDMW), pp:667-674*, 2017.

[2] J. Devlin, M.-W. Chang, K. Lee och K. Toutanova, ”BERT: Pre-training of deep bidirectional transformers for language understanding,” in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*, Minneapolis, 2019.

[3] P. Jaccard, ”The Distribution of the Flora in the Alpine Zone. 1,” *New phytologist*, vol. 11, p. 37–50, 1912.

[4] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer och V. Stoyanov, ”Roberta: A Robustly Optimized BERT Pretraining Approach,” *CoRR*, vol. abs/1907.11692, 2019.

[5] D. Q. Nguyen, T. Vu och A. T. Nguyen, ”BERTweet: A pre-trained language model for English Tweets,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP 2020 - Demos, Online, November 16-20, 2020*, 2020.

[6] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever och others, ”Language Models Are Unsupervised Multitask Learners,” *OpenAI blog*, vol. 1, p. 9, 2019.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser och I. Polosukhin, ”Attention is All you Need,” in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, 2017.

[8] V. Sanh, L. Debut, J. Chaumond och T. Wolf, ”Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter,” i *NeurIPS EMC 2 Workshop*, Vancouver, 2019.

[9] T. Akiba, S. Sano, T. Yanase, T. Ohta och M. Koyama, ”Optuna: A Next-generation Hyperparameter Optimization Framework,” in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[10] A. Z. Broder, ”On the resemblance and containment of documents,” i *Compression and Complexity of SEQUENCES 1997, Positano, Amalfitan Coast, Salerno, Italy, June 11-13, 1997, Proceedings*, 1997.

[11] S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey och N. A. Smith, ”Don't Stop Pretraining: Adapt Language Models to Domains and Tasks,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online, 2020.