TwiCS: Lightweight Entity Mention Detection in Targeted Twitter Streams

Satadisha Saha Bhowmick, Eduard C. Dragut, Weiyi Meng

Abstract—Microblogging sites, like Twitter, continuously generate a large volume of streaming data. This streaming environment creates new challenges for two concomitant Information Extraction tasks: Entity Mention Detection (EMD) and Entity Detection (ED). The new challenges include (1) continuously evolving topics, which may deprecate model-based approaches quickly; (2) non-literary nature of posts, which makes traditional NLP techniques less effective; and (3) huge volume of streaming data, which makes computationally expensive approaches less suitable. In this paper, we propose an approach for EMD/ED whose creation is guided by the constraints specific to streaming environments from the ground up. Our system TwiCS implements this approach. TwiCS employs a computationally light two-phase process. In the first phase, it exploits simple (low computation) syntactic cues to suggest Entity Mention (EM) candidates. In the second phase, it uses occurrence mining to classify candidates according to their likelihood of being true EMs. Our experiments show that TwiCS achieves an average effectiveness improvement of 14.6%, while maintaining at least 2.64 times higher throughput, when compared to several state-of-the-art systems.

1 Introduction

Entity Mention Detection (EMD) is the task of automatically extracting contiguous strings that represent entities of interest in text. In this paper we refer to these excerpts or surface forms (following the terminology in WNUT17 [1]), as Entity Mentions (EMs). In the streaming environment, Entity Detection (ED) aims to cover the breadth of unique entities within text streams, while EMD compiles the mention variations of entities appearing in text. They belong to the broader Named Entity Recognition (NER) problem, and are often the first steps in an NER pipeline.

Example 1. Figure 1 illustrates the variations in which five unique entities are mentioned in tweets: *'trump'*, *'sean spicer'*, *'spicer'*, *'carter page'* and *'cnn'*. They include properly capitalized phrases (e.g., *'Trump'*, *'Sean Spicer'*, *'Spicer'* and *'Carter Page'*), an all capitalized abbreviation (e.g.*'CNN'*), as well as completely non-capitalized mentions (e.g., *'trump'* and *'carter page'*), and a partially capitalized string *'Carter page'*. *'Hold'* is a miscapitalized non-entity string.

In this paper, we study near real-time EMD for Microblog streams, which poses some unique challenges compared with EMD from static or well-formatted text:

- *S. Saha Bhowmick is with the Department of Computer Science, Binghamton University, Binghamton, NY 13905. E-mail: ssahabh1@binghamton.edu*
- *E.C. Dragut is with the Computer and Information Sciences, Temple University, Philadelphia, PA 19022. E-mail:edragut@temple.edu*
- *W. Meng is with the Computer Science Department, Binghamton University, Binghamton, NY 13902. E-mail: meng@cs.binghamton.edu*

Manuscript received October 6, 2020.

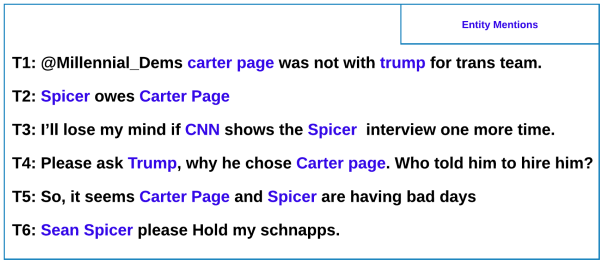


Fig. 1: Entity Mentions in Tweets

(1) Non-Literary Language: Microblog messages tend to exhibit insufficient context, grammar or punctuation errors, or misspellings. Pre-Twitter systems (e.g., [2], [3]) seek understanding of textual semantics through pre-processing like POS tagging and dependency parsing. But the non-literary nature of Microblog text renders models trained exclusively on content with well-formatted grammar and syntax less effective. Hence it has been addressed by a variety of supervised systems [4] developed in this space.
(2) Topical Variation and Evolution: A unique challenge in microblog streams is the coexistence of wide-ranging topics that evolve over time. This brings in an added risk of model deprecation when systems designed on static corpora are applied to streams. Existing EMD systems either refine feature selection for statistical Machine Learning models [4], [2] or build upon sophisticated Deep Neural Networks (DNNs) [5], [6], [7]. These systems are trained off-line with static corpora. To make up for the expansive scope of Microblog stream contents, this requires substantial amount of annotated domain data. Also, the external resources like embeddings set for DNNs or gazetteers used also need to be appropriately updated in order to avoid yielding too many Out-Of-Vocabulary tokens that affect the performance of statically trained models on streaming data.
(3) Volume: Popular streams can generate high volume of messages over an extended period of time. It is imperative for an EMD system operating in this setting to not only detect good quality entities but also to keep up with incoming message traffic. For most supervised systems, especially DNNs, model training and hyperparameter tuning [8] are expensive due to a large number of parameters. This also prolongs test time and increases the delay between message generation and eventual output production. In fact, our experiments showed a state-of-the-art DNN to be 1.83 times slower in EMD than the fastest non-neural alternative.

In stark contrast to existing solutions, we devise a new EMD system called **TwiCS** that requires minimum training, no expensive NLP, and is computationally light. We view

Microblogs as non-static corpora where messages on diverse topics are generated continuously. The streaming nature of this setting, therefore, demands a continuous, incremental solution that can analyze topically evolving messages but with a near real-time processing objective. Our work initiates from the perspective of an EMD system that: (i) is at least as effective as existing state-of-the-art systems, yet (ii) uses only a fraction of their runtimes by being computationally inexpensive. To achieve these somewhat conflicting objectives and address aforementioned challenges, we propose to combine several carefully crafted ideas within a two-phase system. Phase I performs a linguistically shallow and computationally light scan of incoming tweets to quickly detect entity candidates. Phase II extracts more mentions of said entity candidates to validate True Positives (entities) among them. As a departure from once-and-done solutions, Phase II reprocesses a small percentage of tweets (about 5.11% in our empirical studies) with downstream tweets that reveal more information about entity candidates which could not be confidently processed in the former.

Compared to DNNs our solution may appear "simple" but the simplicity is by design. We seek to prove in this study that simple (in the sense of "light") solutions with multiple passes on strategically select inputs, can compete with complex techniques in achieving good effectiveness while significantly outperforming them in throughput, when based on shrewd insights. This appears to be the right strategy computationally to cope with the near real-time constraint of large Microblog streams. We implement TwiCS, an EMD system on Twitter, based on these ideas.

Example 2. We use Figure 1 to help explain the EMD approach of TwiCS. Suppose for this example, Phase I extracts entity candidates based on this heuristic: each capitalized word sequence (except a single capitalized word at the beginning of a sentence) is an EM and suggests a candidate. Suppose the system first encounters T1 in the stream. It ignores *'carter page'* in the initial scan and adds T1 to a queue to revisit later. Next the system encounters T2 and learns that *'Carter Page'* is a true entity and adds it to a knowledge base. At some point, the system decides to revisit T1. This time it decides that *'carter page'* is a valid EM because of its "match" with *'Carter Page'*.

In our experiments, TwiCS is able to achieve an average effectiveness improvement of at least 14.6%, while maintaining higher throughput, at least 2.64 times higher, when compared to single-pass state-of-the-art systems that are built using DNNs or sophisticated NLP techniques for EMD. TwiCS has some additional advantages, such as easy deployment. It can also serve as a strong baseline for comparing EMD solutions for the streaming environment.

We make the following contributions in this paper.

- We propose a novel multi-pass EMD framework with the following key features: (1) employ lightweight computation units better suited for microblog streams, (2) enable collective processing of tweets that leverages occurrence mining to find and label entity mentions, and (3) reprocess a small fraction of tweets with initially ambiguous outputs to get better recall.
- We formulate the Capitalized Shingles (CS) rule to quickly detect entity candidates from text using shallow syntactic cues. We conduct an extensive empirical study to validate its merits.
- Guided by our CS rule, we devise effective, yet efficient, EM candidate extraction and validation algorithms.
- Our extensive experiments show that TwiCS outperforms several state-of-the-art EMD systems in effectiveness, and achieves significantly higher throughput.

The current version of TwiCS with experiments and datasets are available at https://github.com/dalakada/TwiCSv2.

2 Related Work

A majority of EMD systems for tweets are supervised, with either handcrafted linguistic features in a non-neural system or DNNs with minimal feature engineering. The first category of systems like [4], [9] recreate an information extraction pipeline leading up to NER. Twitter NLP [4] first builds a POS tagger and a shallow noun phrase parser, whose outcomes are additional features to a CRF based entity boundary segmentation module. TwitIE [9] customizes several initial components of the open-source GATE ANNIE pipeline, including a tokenizer and a POS Tagger, for the subsequent Named Entity Tagger to better adapt with social media content. Alternatively, Liu et al. [10] proposes a semi-supervised learning model that sequentially applies a K-Nearest Neighbors (KNN) classifier and a linear CRF model to the target tweets. Other recent efforts [11], [12], [13], [14], [15], [16], [17] enhance the feature-set of traditional CRF based models in different ways – with word embeddings and knowledge gathered from gazetteers – to better suit the constraints of microblog paradigm. Also, a variety of systems [18], [19], [20] have adopted neural network based solutions for the sequence labeling task of NER. With systems like [5], [21] we observe the advent of deep learning frameworks to solve the NER problem. The recent WNUT shared tasks delve into an exploration of several NLP tasks including NER specifically catering to the Microblogging domains like Twitter. Both WNUT16 [22] and WNUT17 [1] report multiple systems like [7], [23], [19], [24], [25], [26] applying deep learning specifically for Entity Extraction from Tweets. Therefore, we have chosen Aguilar et al. [6] – a BiLSTM-CNN-CRF architecture that performed best at the WNUT17 task, and BERTweet [27]– a BERT model originally trained on a large Twitter corpus and then finetuned for NER, as two of our strong baselines.

Apart from the aforementioned works, there are also some systems in the literature that promote alternative techniques to NER. TwiNER [28], an example of an unsupervised system, leverages collocation features (from Microsoft Web N-Gram corpus [29]) to generate candidate phrases and then ranks them by exploiting the *"gregarious nature"* of potential entities in a targeted stream.

A few other systems [30], [31] jointly attempt entity extraction with another related Information Extraction subtask. Among the existing industrial systems, OpenCalais [32] performs NER and EL (Entity-Linking) online, on documents submitted as queries. DocTagger [33] implements an end-to-end system interleaving multiple IE tasks like ER, disambiguation and linking, in a shared pipeline.

Two major issues for existing EMD systems are cited in [1]: the lack of annotated data from the Microblog do-

main, and congruently, the difficulty in identifying emerging entity forms. A growing body of work (e.g., [34], [35]) examines cross-domain transferability of DNN features learned from the abundant labelled well-formatted corpora. Handling unseen entity forms however remains a problem that has only been addressed by enhancing architectural robustness of DNNs [36].

TwiCS distinguishes from these systems. It provides a semi-supervised framework that initiates with limited annotation requirements and supports easy appendage of high confidence outputs for quick model retraining. In TwiCS, we posit that better contextualization arrives with targeted streaming of Microblogs [37], [38], where sentences need not be examined in isolation during processing. Our insight is that entity candidate recurrence is typical and should be exploited before delving into more complex solutions. TwiCS relies on incremental occurrence mining that collates EM variations from sentences and analyzes them collectively. We evaluate TwiCS on entity types (e.g., People, Location and Organization) that are predominantly covered in most existing EMD systems. However, TwiCS' EMD capability is not constrained to a particular type set and aims towards a generic Entity Detection objective.

3 System Overview

TwiCS follows a continuous and iterative process for entity detection. To this effect, it employs a multi-pass approach, that scans through tweets to accumulate shallow syntactic features of the constituent candidates. Tweets are retained in the system until all candidate mentions in them are confidently detected. This is a significant departure from the once-and-done approach employed by existing systems. We argue that tweets introduced at different processing time points can mutually benefit from knowledge aggregated over multiple (often non-consecutive) iterations. Keeping in mind the possible computational overhead of a multi-pass approach, we intentionally avoid any expensive in-depth semantic analysis and employ computationally light methods. Furthermore, we do not process all messages in a stream multiple times. In fact, only a very small percentage of them (less than 6% in our experiments) need to be reprocessed after the first iteration. We now describe the building hypotheses and architecture of TwiCS.

3.1 Building Hypotheses

We construct our solution based on several hypotheses. We categorize them as EM extraction [E-Hx] and EM computation [C-Hx]. A discussion on the validity of these hypotheses will be presented through analyses in later sections.

E-H1: Messages in targeted streams tend to congregate in topics or conversations each of which contains a small set of entities.

E-H2: Entities appear in limited number of appellative forms that are progressively discovered as more messages on the same topic are processed.

E-H3: Although nonliterary language is endemic to microblogs, EMs typically regress to proper orthography in messages (e.g., capitalization in English and many other languages).

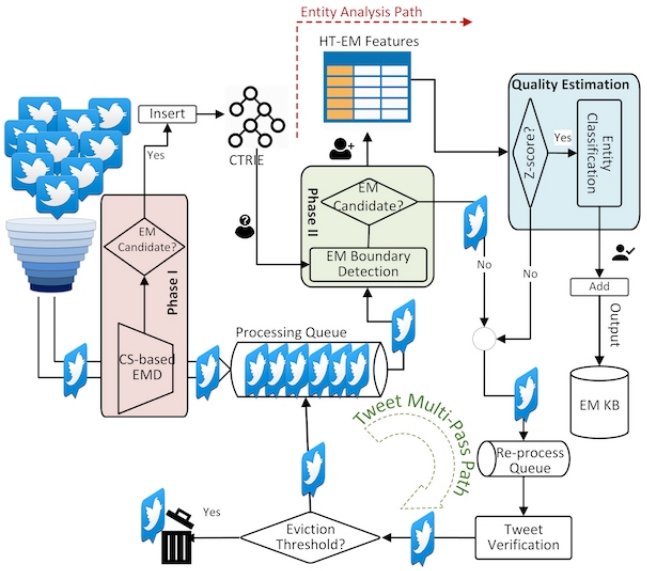


Fig. 2: Multi-pass Entity Mention Detection System

C-H1: (Base Case) Majority of EMs can be detected with simple, computationally inexpensive syntactic rules.

C-H2: (Step Case) EMs in messages with language misuse may not be extracted the first time they are seen, but they can be confidently extracted if revisited.

These hypotheses are the guiding principles in building TwiCS: (1) guided by E-H1, we form groups of tweets with similar topics or contexts (a principle shared with other NER tools, e.g., [28], [39]); for Twitter, conversation streams usually circulate using Hashtags which can also be equated with topic-specific identifiers for topical clustering from streams. (2) guided by E-H2 and E-H3, we leverage shallow linguistic features (like capitalization and punctuation usage) to quickly extract many EMs in one pass; (3) guided by C-H1, we seed the detection of more EMs with the EMs discovered according to (2); (4) guided by E-H2, we iteratively grow the set of known EM variants for an entity, and (5) guided by C-H2, we strategically revisit tweets whose EMs were not detected in previous iterations. Our empirical studies reveal that the Base Case (C-H1) helps extract about 55% EMs on average while the Step Case (C-H2) covers 43% additional EMs on average. We elaborate on empirical validation of these hypotheses in Section 4.2.

We use the example in Figure 1 to explain how E-Hx and C-Hx guide EMD computation in TwiCS. Suppose that TwiCS first meets T1. It ignores *'carter page'* as it is not confident if it is an EM, but it adds T1 to a queue to revisit later. Suppose that next it encounters T2 which has *'Carter Page'*. TwiCS supposes that this can be an entity (E-H3) and adds it to the seed set (C-H1, base case). For T4 next, it is able to recognize *'Carter page'* via a "match" with *'Carter Page'* in the seed set (E-H2). It adds *'Carter page'* to its knowledge base. With T5, TwiCS is confident that *'Carter page'* is a true entity. At some point, TwiCS decides to revisit queued tweets and re-processes T1 (C-H2, Step Case). It decides that *'carter page'* is also a valid EM because of its "match" with *'Carter Page'* (E-H2) and adds it to its knowledge base.

TwiCS follows the principle of shallow syntactic analysis to design entity extraction features. While this paper focuses on messages in English, orthographic distinction of proper

nouns is common for a variety of languages [40], including many European languages (e.g., Spanish, French, Italian, Turkish). So, the shallow syntactic element of TwiCS can be easily extended to accommodate many other languages.

3.2 Architecture

Figure 2 illustrates the overall architecture of the TwiCS system. This framework facilitates continuous execution of a tweet stream over multiple iterations (or batches). A batch is a set of tweets on the same topic that will be processed together. A single processing cycle of an incoming batch of tweets can be divided into the following steps:
(1) Streaming module fetches a stream of tweets, on a particular topic, using the Twitter streaming API.
(2) Phase I (Base Case) extracts candidate (seed) entities from these tweets and forwards them to the Active Processing Queue (APQ).
(3) Phase II (Step Case) recovers mentions of seed entities from tweets in APQ in a syntax agnostic fashion. It populates a frequency distribution over the syntactic variations of mentions per entity.
(4) The cumulative frequencies in the resulting distribution are sorted by some scores (Z-scores) and candidates with scores above a pre-defined threshold are classified, according to their likelihood of being a true entity. Those whose scores do not cross the threshold are retained for further processing.
(5) Candidate database is updated with classification labels. Candidates that are consistently labelled as 'entity' or 'non-entity', across several iterations, are stored as learned knowledge, in an internal Candidate Knowledge Base.
(6) Tweets in the current iteration go through a verification process using the learned labels of candidates. Tweets in which every candidate mention has been confidently labelled based on certain completion criteria are transferred to an Output Queue.
(7) Tweets that do not satisfy the completion criteria are retained in a Reprocess Queue (RQ). They can later be selected to be moved to the APQ for re-evaluation or removed from the system based on an eviction strategy. As will be shown by experiments later, only a small percentage of tweets (less than 6%) needs to be reprocessed.

We assume that a topic filtering step extracts targeted streams from Twitter before they are ingested to TwiCS. This is common practice in Twitter monitoring [39], [1].

We elaborate on these steps in the following sections.

4 Phase I: Shallow Entity Detection

We describe Phase I according to hypothesis C-H1 in this section. We provide a quantitative analysis to support E-H2 and E-H3, which allows us to devise a light computational unit in TwiCS. This is the only component of TwiCS that needs to be customized to accommodate other languages. It is computationally inexpensive and requires no supervision.

4.1 Objectives

This phase is based on the *Capitalized Shingles (or CS)* heuristic, and some additional considerations. The primary objectives of this phase are two-fold. For tweets in each topic: (1) generate an initial set of seed entity candidates, and (2) collect syntactic features for individual entity mentions during candidate extraction.

We also remove stopwords from individual tweets in the first pass, except for those appearing in quoted strings. We also allow singular occurrences of select prepositions and/or articles within a CS instance. This ensures that strings like *State University of New York* can be correctly recognized as a single CS instance.

4.2 Exploiting Regression to Proper Orthography

We give supporting evidence for hypotheses E-H2 and E-H3, that help implement C-H1 – the basis of Phase I, in this section. We conduct a study on five stream datasets (D_1-D_5) and one randomly-sampled third party dataset WNUT17, all to be described in detail in Section 8. They amass over 1M tweets from a wide array of topics (e.g., Sports, Politics, Entertainment). Such topic diversity ensures an unbiased assessment of our hypotheses. We also analyze to what degree these hypotheses are validated in streaming and non-streaming environments. We now define the concept of Capitalized Shingle. A **capitalized word** is a word whose first character is in upper case.

Definition 1. (***Capitalized Shingle***) A Capitalized Shingle (CS) is the longest possible consecutive sequence of capitalized words with the following three exceptions: (1) If such a sequence starts a sentence, it must contain at least two capitalized words for it to be a CS. (2) Each such sequence is allowed to contain a singular occurrence of a preposition and/or article between otherwise consecutive capitalized words. (3) If every word in a sentence is a capitalized word, no CS is generated for this sentence.

Figure 6a highlights the CS instances in Figure 1 according to this definition. The convention for representing entities in English is to use capitalized word sequence. Although not all Twitter users consistently follow this, a significant fraction of them do. But in English a sentence also begins with a capitalized word. Thus, when the first word of a sentence is not immediately followed by (except a singular occurrence of preposition and/or article) other capitalized word(s), it might not represent an entity by itself. This explains the first exception. Through the second exception we allow single non-capitalized instance of certain prepositions and/or articles within capitalized words, together forming a CS. For example, in the sentence *'Manchester by the Sea won best picture'*, the exception allows the movie name *'Manchester by the Sea'* to be detected as a CS. Finally, when every word of a sentence is capitalized, it is more indicative of a writing style than highlighting entities.

Supporting E-H2. We first examine the various orthographic patterns in which labelled entity mentions appear in the annotated datasets. As reported in Table 1, these mentions can be exhaustively enumerated into six different surface forms, thereby validating the assumption in E-H2. We elaborate on these individual forms in Section 5.3. Furthermore, we find that this hypothesis holds both in the case of streaming datasets that we had collected for experimentation as well as the randomly sampled tweets in the WNUT17 dataset.

Supporting C-H1 through E-H3. The findings in Table 1 reveal that the two capitalized surface form variants

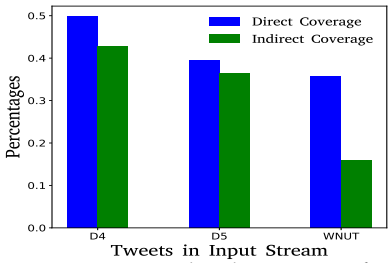


Fig. 3: Tweet level coverage of CS

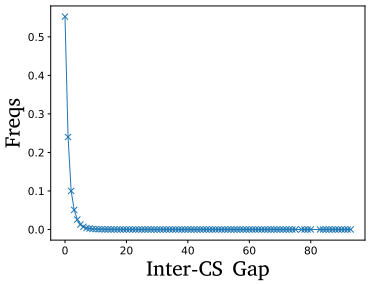


Fig. 4: CS Inter-arrival Distribution

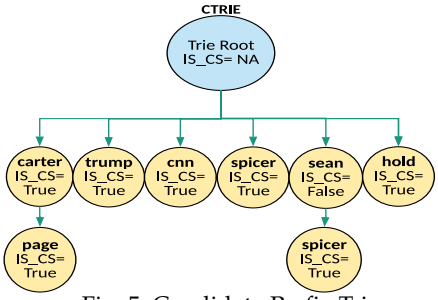


Fig. 5: Candidate Prefix Trie

TABLE 1: Percentages of different appellative forms of annotated EMs. Acronym Legend: Cap = Proper Capitalization; Sub Cap = Substring Capitalized; s-o-s Cap = Start-of-sentence Capitalization; full Cap = Fully Capitalized; non Cap = Non Capitalized; non-Disc = Non Discriminating

Dataset	Cap	Sub Cap	s-o-s Cap	full Cap	non Cap	non-Disc
D_1–D_3	68.2	1.4	7.1	12.4	3.5	7.4
WNUT17	63.8	7.6	2.4	6.5	17.1	2.5

cumulatively cover a majority of EMs thereby validating E-H3. For the streaming datasets D_1 through D_3, about 80.6% of entity mentions appear in these two forms. Our initial hypothesis expected datasets collected from tweet streams having high volume topically related content to exhibit regression to proper entity orthography. However, as is evident from Table 1, even WNUT17, essentially a random sampling of content from the Twitterverse, with little topical integrity, preserves E-H3. Specifically, 70.3% of entity mentions in WNUT17 appear in some manner of capitalization. However, the percentage is indeed lower than the case of streams. Finally, this also provides a clear intuition to exploit these orthographic patterns and ensure a high coverage of entities during the candidate extraction phase, as mentioned in C-H1. To this end, we propose the CS Heuristic as the basis for entity candidate extraction:

CS Heuristic: Each CS instance is an Entity Candidate.

Characterizing Datasets through Orthographic Regularity. Now with a clear motivation for orthography driven candidate extraction, we gather some statistics to gauge its effectiveness. While the CS Heuristic we propose is intended for tweet steams, these statistics help us characterize and distinguish voluminous streams from third-party benchmarking datasets like WNUT17 which collates a random sampling of tweets. It also helps illustrate to what degree the CS Heuristic can capture entity candidate in streaming vs non-streaming Twitter datasets.

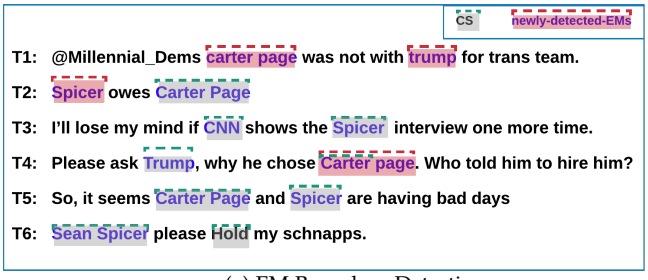
We first estimate the tweet-level coverage of CS on datasets D_4, D_5 and WNUT17 using two notions. **Direct coverage** reflects the percentage of tweets having one or more CS instances. **Indirect coverage** is the percentage of tweets that have no CS instance but contain a string that matches with a CS instance in another tweet when case or capitalization is ignored (e.g., "trump" matches with "Trump"). Note that CS instances or strings that match with CS instances can all refer to mentions of entities. As illustrated in Figure 3, a considerable fraction of tweets in streams contains either CS (between 40% - 50%) or strings that match with CS (between 35% - 40%). Together they generate EM candidates through a hefty coverage of 75-90% of tweets. These strongly motivate our approach. For WNUT17 though, the direct coverage is lower, at 35.7% which excludes a sizeable fraction of tweets during candidate extraction. However the indirect coverage for WNUT17 is lower still at 15.8% showing that a random sampling of tweets do not capture the innate tendency of tweet streams to repeat entity mentions. Since the components of TwiCS following Phase 1 capitalize on this specific trait, this limits its overall EMD effectiveness on WNUT17.

Computationally, our approach is useful if CS instances appear frequently and with a certain degree of regularity. We introduce the notion of **Inter-CS gap**, which is the number of tweets that enter the stream between two successive CS occurrences. We plot the CS inter-arrival distribution over different gap values encountered within a stream in Figure 4 using the dataset D_5. The distribution closely resembles a negative exponential function of the form: $f(x) = ae^{bx}$, where the properties of the best fitted curve are a = 0.552, b =-0.827 at an SSE of 0.00012, with mean μ=0.907 and standard deviation σ= 1.635. The distribution is heavily skewed towards lower gap values, indicating that CS instances appear fairly frequently within a stream: almost every other tweet in a stream has an CS instance. This also strongly endorses an approach that capitalizes on the occurrences of CS instances. Unlike D_5, WNUT17 is a random sample of tweets that are not timestamped or temporally ordered. Therefore the concept of Inter-CS gap does not apply to WNUT17.

Furthermore, the considerable indirect coverage encourages us to devise a syntax agnostic method in Phase II, and capture additional mentions of CS candidates. The classifier used to determine potential entities from the candidate occurrence distributions will further improve confidence yielded by using CS alone. We now present the Entity Candidate Extraction module of TwiCS, guided by the CS Heuristic, and its associated data structures.

4.3 *CS+*: CS based Extraction Module

Our CS-based Shallow Entity Detection module considers individual tweet sentences as input but discards the ones entirely in uppercase or lowercase, because they cannot contain CS. It also discards sentences where every word is a capitalized word, as they cannot contain any meaningful CS. A basic sentence-splitter and tokenizer is used in the preprocessing step to split a tweet into sentences and remove common stopwords from them. Phase I initializes two data structures, **CandidatePrefixTrie** and **TweetBase**, which are utilized and modified in the later steps of TwiCS.



(a) EM Boundary Detection

candidate	length	cap	substring cap	s-o-s cap	all-cap	non-cap	non discriminative	cumulaive	label
trump	1	159	1	11	3	9	18	201	entity
carter page	2	221	1	0	0	6	41	269	entity
hold	1	2	0	0	0	9	1	12	non entity
sean spicer	2	10	0	0	0	0	4	14	entity
spicer	1	1	0	3	0	1	0	5	ambiguous
cnn	1	1	0	0	0	0	0	1	na

(b) Candidate Syntactic Distribution

Fig. 6: Syntax Agnostic Mention Detection for Quality Estimation

We construct a prefix Trie forest, referred to as **CandidatePrefixTrie** or **CTrie**, for efficient indexing and storage of entity candidates extracted in Phase I. CTrie functions like a regular Trie forest with individual nodes corresponding to a token in a candidate entity string. A **path** is a sequence of nodes starting from the root that forms an entire entity candidate string. Entity candidates with overlapping prefixes are part of the same Trie in the forest. Nodes in the Trie, which correspond to the last token of extracted candidates, contain: (a) a *Flag* set to True indicating the path from root to this node represents a prefix that is a CS instance, found in Phase I and (b) additional information about syntactic features of the candidate string (e.g., length). They are gathered during Phase I from the contexts of its occurrence in many tweets. Entity candidates stored in CTrie are all in lower case format. While hash table is a reasonable alternative data structure for storing candidate entity strings, CTrie better facilitates the boundary segmentation algorithm in Phase II (as explained later) and is thus preferred.

Every record in **TweetBase** consists of individual tweet sentences indexed by both tweet ID and sentence ID, and an entity candidate list to be updated after each phase.

In addition to generating entity candidates using the CS Heuristic, we pass them through a series of filters. Each filter considers additional syntactic features and removes CS instances that are unlikely true entities. These filters include:

(1) Digit Processing: Standalone numbers are not regarded as entity candidates. Numeric tokens are allowed only if they are part of a longer CS candidate.

(2) Slang Check: Removes commonly used slangs.

(3) Length Check: Since entities are typically short, we do not allow candidates longer than 6 tokens and discard single character candidates, which are unlikely to be entities.

Misspelled entities can still be recognized as candidates separately and detected through the rest of the pi like their intended counterparts. We plan to include clustering of misspelled entities with the actual ones in future work.

Example 3. We use the set of tweets from Figure 1 to demonstrate how CS+ contributes to the EMD process, along with modifying the internal data structures. CS+ is able to detect all entities within the tweets of Figure 1 as potential candidates: *'Trump'*, *'CNN'*, *'Spicer'*, *'Carter Page'* and *'Sean Spicer'*. However, it failed to detect several entity mentions: *'Spicer'* in T2 owing to Exception 1, *'carter page'* and *'trump'* in T1; it only detected part of another mention of *'Carter page'* in T4, and also wrongly recognized *Hold* as an entity candidate. In Section 5, we introduce Phase II that aims to correct these failures in detecting entity mentions.

5 Phase II: EM Boundary Detection

In this phase (the Step Case), the primary goal is to identify additional EMs of existing entity candidates by devising a procedure that overcomes the reliance of Phase I on syntactic features. This phase aims to collect all possible candidate mention variations of the seed entities in a tweet, *irrespective of their capitalization patterns*. This helps improve the recall. To improve precision, this phase constructs a frequency distribution over the different syntactic variations of each candidate entity and utilizes it to distinguish entities from non-entity strings.

5.1 Objectives

We have the following specific objectives in this phase:

1. Removal of False Negatives: False Negatives happen in Phase I when true EMs are not valid CS instances, mostly because of being improperly capitalized or single word entities appearing at the start of a tweet sentence. For example, in Figure 1, *carter page* and *trump* in T1 and *Spicer* in T2 are false negatives.

2. Removal of False Positives: Misleading capitalizations may result in non-entity phrases to be extracted as entity candidates in Phase I. For example, *Hold* in T6 in Figure 1 is a false positive.

3. Correction of Partial Extraction: Partial extractions happen when only part of an entity string is extracted. Such a phenomenon is usually caused by mis-capitalization of multi-word entities. In T4 in Figure 1, entity mention *Carter page* was only partially extracted in Phase I. Correcting such partial extractions improves both recall and precision of tweet-level EM detection.

5.2 EM Boundary Segmentation

Some traditional NER systems use CRF (conditional random field) to segment and identify EM boundaries in text (e.g., [2], [10]). They require large feature spaces that include rich token tagging and parsing. These pre-processing steps are computationally expensive. Since we deliberately constrain TwiCS to not use them and be computationally light, we face different challenges in EM segmentation.

In Phase I, recognizing an EM is pre-conditioned to the capitalization consistency exhibited by each of its tokens. In Phase II, we solve the boundary problem by interpreting tweet tokens without any reliance on syntactic cues. We analyze a token in conjunction with CTrie. We encounter three types of tokens when traversing CTrie:

(i) A token whose first letter is capitalized.

(ii) A token that is not capitalized, but matches a candidate node on the current CTrie path, when cases are ignored.

(iii) Non-capitalized token matching no node in current path. **The problem** is to determine whether a token corresponds to an entity candidate, either alone or together with up to k tokens following it. Note that the case of partial capitalization can appear both in the prefix or suffix of a candidate. Phase II checks every tweet t in TweetBase and identifies the set of longest possible subsequences that match existing entries in the CTrie, while case is ignored (e.g., "sean" is a match for "Sean"). With the matches hereby encountered, Phase II constructs a frequency distribution over the different mention variations of each entity candidate. As a consequence, candidate mentions extracted during Phase I are verified, and sometimes corrected. For example, CS guided extraction for T4 can only find the partial EM, 'Carter' in Figure 1 which stands corrected to the complete mention, 'Carter page' in Figure 6a. Algorithm 1 compiles the various considerations of this module. The resulting process is syntax agnostic. It initiates a window to incrementally go through a sequence of consecutive tokens. In each step it checks:

a) whether the subsequence within the current scan window corresponds to an existing path in the CTrie (lines 13-24). If true, it implies that the search can continue along the same path, by including the next token to the right within the window in the next iteration.

b) whether the node corresponding to the last token of the subsequence, has its *flag* set to True (lines 15-18). This implies that the subsequence is the current longest match with a candidate found in Phase I.

Note that the subsequence match is performed without checking any immediate syntactic evidence of constituent tokens. This renders the aforementioned scenarios (i) and (ii) to be equivalent in terms of Algorithm 1. In case of a mismatch, i.e., scenario (iii), the module stores the last recorded matching subsequence within the current window, and then skips ahead by initializing a new window from the position right after the last matched subsequence. The search for a new matching subsequence path is initiated from the root of the CTrie. However, if the last search had failed to yield a match with any of the existing candidates in CTrie, the new window is initialized from a position that is to the immediate right of the first token in the previous window (lines 32-35). The process is repeated until all tokens in the sequence are consumed.

Example 4. Consider T2 in Figure 6a. The CTrie, at the end of Phase I, has both *'spicer'* and *'carter page'* as candidates (Figure 5). The window initiates scan from the first token *'Spicer'*. The algorithm traces the path from root to the node having *'spicer'* and recognizes this instance of the candidate. Appending the ensuing token *'owes'* to *'Spicer'* fails to find a match in the CTrie. Thus, it stores *'spicer'* as the previous longest match, thereby detecting this mention missed earlier during Phase I, and resets both the window and the CTrie traversal from the mismatched token. Since no other path from the root begins with the node *'owes'*, another mismatch will reset search from the next token, *'Carter'*. Tracing the path in CTrie starting from the node *'carter'*, the module matches this token and subsequently finds *'Carter Page'* as the longest matching subsequence, along this path.

Algorithm 1: Boundary Segmentation

input : sequence of consecutive tokens with positions $sequence$, CandidatePrefixTrie $CTrie$
output: candidates extracted, $EM\_Candidate\_List$

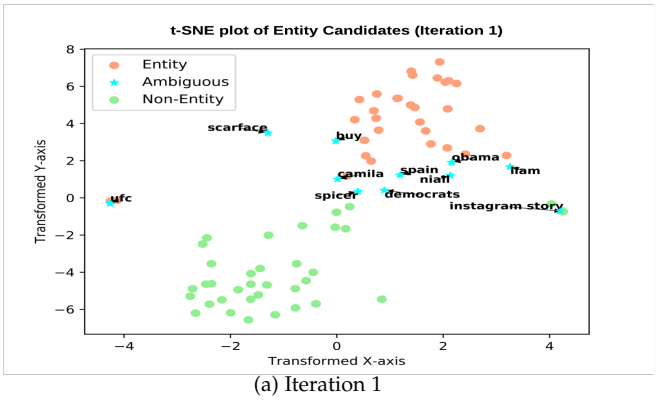
```
1  left ← 0; right ← 0; start_node ← CTrie;
2  last_cand ← "NAN";last_cand_substr ← "";
3  reset ← False; last_cand_pos ← empty tuple;
   EM_Candidate_List ← [];
4  while right < len(sequence) do
5      if reset then
6          start_node = CTrie;
7          last_cand_substr = "";
8          last_cand_pos ← empty tuple;
9          left = right;
10     curr = sequence[right];
11     last_cand_sequence = sequence[left:right];
12     cand_str = last_cand_substr+curr;
13     if curr ∈ start_node.path.keys() then
14         reset = False;
15         if start_node.path[curr].flag then
16             last_cand= cand_str;
17             //position of last matched subsequence;
18             last_cand_pos ← (left,right);
19         else
20             if ((right==len(sequence)) &
                (last_cand=="NAN") &(left < right))
                then
21                 right = left;
22                 reset = True;
23         start_node=start_node.path[curr];
24         last_cand_substr=cand_str;
25     else
26         if last_cand ≠ "NAN" then
27             EM_Candidate_List.append(last_cand);
28             last_cand = "NAN";
29             //restarts scan after last matched
                subsequence;
30             right= last_cand_pos[1];
31             reset = True;
32         else
33             if left < right then
34                 right = left;
35             reset = True;
36     right+=1;
37 end
38 if last_cand ≠ "NAN" then
39     EM_Candidate_List.append(last_cand);
   return : EM_Candidate_List
```

5.3 Syntactic Distribution of Candidates

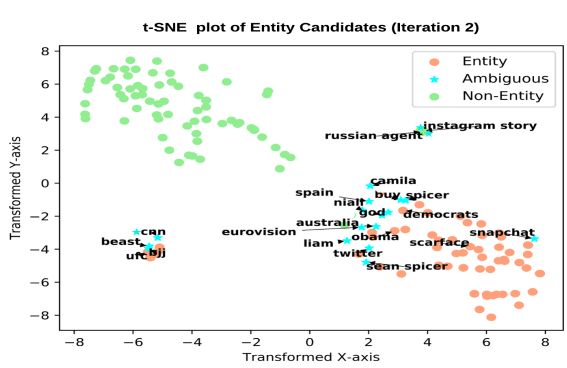
An important task of Phase II is to construct the syntactic distribution of each entity candidate, over its different mention variations encountered in the tweets processed thus far (Figure 6b). Phase II enumerates six syntactic variations of each entity candidate and computes their corresponding frequencies (E-H2).

(1) **Proper Capitalization**: This corresponds to a CS.

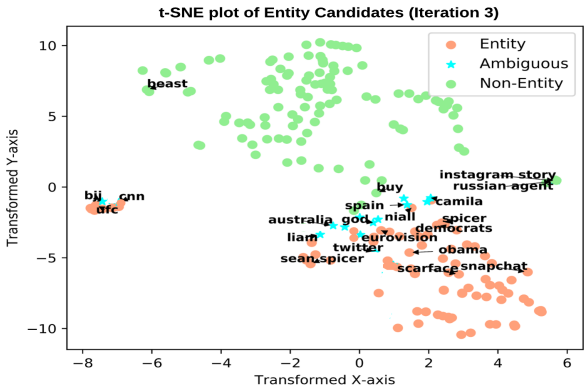
(2) **Start-of-sentence capitalization**: A single word (unigram) is capitalized at the start of sentence.



(a) Iteration 1



(b) Iteration 2



(c) Iteration 3

End-of-Iteration	Tweet Sentences	Processing State
1	@Millennial_Dems carter page was not with trump for trans team.	Complete
	Spicer owes Carter Page	Incomplete
	I'll lose my mind if CNN shows the Spicer interview one more time.	Incomplete
	Please ask Trump, why he chose Carter page. Who told him to hire him?	Complete
	So, it seems Carter Page and Spicer are having bad days	Incomplete
	Sean Spicer please Hold my schnapps.	Complete
2	Spicer owes Carter Page	Incomplete
	I'll lose my mind CNN shows the Spicer interview one more time	Incomplete
	So, it seems Carter Page and Spicer are having bad days	Incomplete
3	Spicer owes Carter Page	Complete
	I'll lose my mind CNN shows the Spicer interview one more time.	Complete
	So, it seems Carter Page and Spicer are having bad days	Complete

(d) Multipass Tweet Processing

Fig. 7: Multi-pass Processing of Ambiguous Candidates and Incomplete Tweets

(3) **Substring capitalization**: Only a proper substring of a multi-gram candidate is capitalized.
(4) **Full capitalization:** Abbreviations like *'UN'* or *'USA'* where the entire string is capitalized.
(5) **No capitalization**: The entire string is in lower case.
(6) **Non-discriminative**: A sentence in full upper or lower case, or with first character of every word capitalized is not informative for classification, even if a candidate is found.

These frequencies form the syntactic distribution over mention variations of each seed entity candidate. The distribution is subsequently used to predict the likelihood of a candidate being a true entity. We use a hash table like data structure, called **CandidateMentionBase** (CB), to update these frequencies during the execution of Phase II.

6 Candidate Quality Estimation

We use the frequency features collected in **CandidateMentionBase** at the end of Phase II to learn a candidate quality function (*CQ(candidate)*), that determines the likelihood of a candidate being true positive. We use semi-supervised learning to train an SVM classifier [41] (with an RBF kernel and L2 regularization) on dataset D_4. We periodically expand the training set with test cases that produce high-confidence outputs, where class labels (e.g., 'entity' or 'non-entity') remain unchanged for five consecutive processing cycles. The quality score is interpreted as the probability of a candidate being a true entity. It is divided into three empirically determined ranges– from variation in performance over different values on our entire annotated data (D_1-D_3) – each corresponding to a class label:
(1) α: $\geq$0.8, candidate is confidently labelled as an **entity**.
(2) β: $\leq$0.4, candidate is confidently labelled as a **non-entity**.
(3) γ: ϵ(0.4, 0.8), candidate is deemed **ambiguous** and retained in RQ for future computation cycles.

A candidate's frequency distribution over mention variations is more reliable when the frequencies are collected over more than just a few occurrences (avoiding randomness). Consequently, the classifier also performs better in distinguishing entities from non-entity strings for frequent candidates. Thus, it is important to classify candidates when their cumulative frequency over all mention variations is sufficiently high. Since the absolute cumulative frequency of candidates depends on the number of tweets in a discussion stream, we use Z-score [42], [43] to achieve normalization. Theoretically, Z-score indicates how many standard deviations apart the candidate frequency is from the expected value (or mean) of the distribution. For a distribution with mean μ and standard deviation σ, the Z-score is given as $Z = \frac{freq-\mu}{\sigma}$. Only candidates with Z-score higher than an optimal threshold are sent to the classifier for Quality Estimation. For this reason, the classification label of the candidate *'CNN'* in Figure 6b is not available ('na') and does not appear in Figure 7a, even though it was discovered.

We performed a t-SNE [44] based transformation of the candidate feature space for visualization. Figures 7a to 7c reveal an interesting behavior – the ambiguous candidates separate data points from the 'Entity' and 'Non-Entity' classes, much like a decision boundary between them.

7 Verification of Tweets and Eviction

Once the candidate quality scores are obtained from the classifier, the TweetBase is checked to find tweets that have been completely processed, i.e., their Phase II EM candidates have all been confidently labelled by the classifier (classification score is in range α or β). Note that any candidate that receives a score in the ambiguous range γ requires further

TABLE 2: Twitter Datasets

Dataset	Size	#Topics	#Hashtags	#Entities/TweetSet
D_1	1K	1	1	283
D_2	3K	3	6	906
D_3	6K	5	5	674
D_4	38K	1	1	≈7000
D_5	1.035M	3	23	-
D_6	10.26M	-	-	-
WNUT17	1287	-	-	-
BTC	9553	-	-	-

syntactic evidence to be successfully classified in later processing cycles. Tweets containing ambiguous candidates are tagged "incomplete" while the rest are tagged "complete" at the end of an iteration. Tweet sentences that tagged "complete" are evicted from the TweetBase and appended to an *Output Queue (OQ)*. Figure 7 illustrates how incomplete sentences are completed through different iterations.

Incomplete tweets remain in the *Reprocess Queue* (RQ) so that they can be selectively included into the *Immediate Processing Queue (IPQ)* of Phase II, and processed in subsequent iterations. Such a mechanism converts the Phase II processes and associated data-structures into stateful computational units, that maintain and update their execution states in every processing cycle. The CTrie is also a stateful data structure as it stores newly discovered candidates from each iteration and expands the Prefix Trie accordingly. However, the Shallow Entity Detection module of Phase I is stateless. It processes each tweet only once during its lifecycle within the system, in the order of its entry into the processing cycle.

TwiCS also updates a list of entities and non-entities, labelled during each iteration. Reusing previously learned candidates helps to accelerate processing of later tweets.

Example 5. Over three successive iterations, Figures 7a to 7c demonstrate how various ambiguous candidates from earlier iterations get disambiguated (confidently classified) in later ones. In each figure, we label the candidates that remain ambiguous in the current iteration, along with the ones that were previously ambiguous, but have just been disambiguated (as an entity or non-entity). Consider ambiguous candidates *'Spicer'* and *'CNN'* in the first and second iterations (Figure 7a and 7b; *'CNN'* is actually not in Figure 7a due to its low Z-score in iteration 1), respectively. They are eventually classified as entities in the third iteration (Figure 7c), after repeated processing attempts over a course of three iterations. Figure 7d shows how processing states of incomplete tweets (from iterations 1 and 2) simultaneously change with the disambiguation process of their constituent ambiguous candidates, over multiple iterations.

The multi-pass framework of TwiCS retains incomplete tweets in RQ until confident classification of all of their EM candidates. However, further processing of incomplete tweets, after a certain number of iterations, has diminishing returns in improving effectiveness, while piling up processing overhead. To remedy this, we limit retention of incomplete tweets to only a small number (ep) of additional processing cycles since entry, before they are evicted. In the current version of TwiCS, $ep = 2$. We leave the investigation of more performance optimizing and memory efficient eviction techniques for future work.

8 Experiments

We conducted extensive experiments to evaluate the effectiveness and efficiency of TwiCS for Entity Mention Detection in tweets. We implement our algorithms in Python 3.5 and execute them on a system with 3.60GHz Intel Core(TM) i7-4790 CPU and 8GB memory. The Deep learning models were executed on a NVIDIA Tesla T4 GPU on Google Colaboratory. Datasets and pieces of software used for our experiments are available at Github.

TABLE 3: Evaluating EMD on Third Party Datasets

Dataset	System	Precision	Recall	F1
WNUT17	Aguilar et al.	0.73	0.42	0.53
	BERTweet	0.63	0.46	0.53
	TwiCS	0.497	0.50	0.50
BTC	Aguilar et al.	0.77	0.40	0.53
	BERTweet	0.66	0.59	0.62
	TwiCS	0.64	0.54	0.59

Baselines: We compare TwiCS with six state-of-the-art systems: BERTweet [27], Aguilar et al. [6], Stanford NER [2], Twitter NLP [4], OpenCalais [32], and TwiNER [28]. BERTweet, Aguilar et al., Stanford NER and Twitter NLP are supervised, with the first two being Deep Learning architectures; TwiNER is unsupervised and performs EMD (without type classification) in an offline setting; OpenCalais is an industrial system operating in an online setting. We were unable to compare TwiNER's performance against TwiCS at scale because of its reliance on Microsoft Web NGram, that supports limited API calls.

Datasets: We use a combination of third-party datasets, along with the ones curated (in Table 2) by crawling actual message streams from Twitter, to evaluate both EMD effectiveness and efficiency. We follow the principle of focused crawling in data collection so that they reflect topical coherence (recall hypothesis E-H1). The topics covered include Politics, Sports, Entertainment, Health and Science. This helps to preserve the naturally occurring topic-specificity of tweet streams, that we exploit for candidate extraction, without making the analysis biased towards a particular topic. In practice, a topic classifier [45] is to preface an NER tool for streams. We seed the search with a set of top trending hashtags and iteratively enlarge the dataset by submitting queries with frequent n-grams and new hashtags.

We begin our effectiveness evaluation with two popular third-party datasets **WNUT17** [1] and **BTC** (Broad Twitter Corpus) [46]. In order to test recognition of novel and emerging entities, these datasets collect content from different social media platforms, with little adherence to specific events and/or topics. WNUT17 draws tweets from time periods of recent disasters like the Rigopiano avalanche and the Palm Sunday shootings and samples from Reddit, YouTube and StackExchange comments. BTC dataset curates a general collection of tweets out of which only 200 adheres to the MH17 disaster. Rest are randomly sampled. Hence these datasets are very different from a typical microblog streaming dataset. Their low direct and indirect coverage of entity mentions do not conform with the E-H1 hypothesis. Despite extensive search we could not find third-party datasets directly accommodating targeted streams. However it is still interesting to compare the performance of TwiCS with other baselines on WNUT17 and BTC.

TwiCS's results on third-party datasets are shown in Table 3. When computing performance we only consider EM (called *surface form* in [1]) detection without the entity type information. To compensate for these datasets' low direct

TABLE 4: Effectiveness of TwiCS vs. state-of-the-art

Dataset↓	Task →	EMD			ED		
	System ↓	P	R	F1	P	R	F1
D1	TwiCS	**0.83**(0.81)	**0.79**(0.77)	**0.8**(0.79)	**0.79**(0.76)	**0.72**(0.68)	**0.75**(0.72)
	Aguilar et al.	0.76(0.72)	0.58(0.61)	0.66(0.66)	0.76(0.72)	0.55(0.52)	0.64(0.60)
	BERTweet	0.66(0.65)	0.49(0.50)	0.56(0.57)	0.42(0.43)	0.50(0.51)	0.46(0.47)
	Twitter NLP	0.65(0.65)	0.47(0.49)	0.55(0.56)	0.64(0.60)	0.51(0.48)	0.57(0.54)
	Stanford NER	0.66(0.67)	0.30(0.31)	0.41(0.42)	0.74(0.74)	0.31(0.35)	0.43(0.48)
	OpenCalais	0.49(0.38)	0.48(0.40)	0.48(0.39)	0.59(0.54)	0.37(0.35)	0.45(0.42)
D2	TwiCS	**0.81** (0.73)	**0.76** (0.73)	**0.79**(0.73)	**0.65** (0.61)	**0.63** (0.61)	**0.64** (0.61)
	Aguilar et al.	0.78 (0.77)	0.64 (0.62)	0.70 (0.68)	**0.65** (0.62)	0.59 (0.60)	0.62 (0.61)
	BERTweet	0.77(0.72)	0.63(0.65)	0.69(0.68)	0.5(0.52)	0.63(0.62)	0.56(0.57)
	Twitter NLP	0.77 (0.76)	0.48 (0.50)	0.59 (0.59)	0.64 (0.63)	0.51 (0.53)	0.56 (0.58)
	Stanford NER	0.79 (0.79)	0.41 (0.43)	0.54 (0.56)	**0.69** (0.69)	0.36 (0.38)	0.47 (0.49)
	OpenCalais	0.62 (0.62)	0.32 (0.30)	0.42 (0.40)	0.55 (0.53)	0.31 (0.33)	0.39 (0.40)
D3	TwiCS	**0.84**(0.77)	**0.80**(0.84)	**0.82**(0.8)	0.62(0.59)	**0.63**(0.63)	**0.63**(0.61)
	Aguilar et al.	**0.84**(0.76)	0.63(0.66)	0.72(0.70)	**0.63**(0.61)	0.61(0.63)	0.62(0.62)
	BERTweet	0.70(0.68)	0.54(0.58)	0.59(0.63)	0.53(0.57)	0.55(0.52)	0.54(0.55)
	Twitter NLP	0.67(0.59)	0.41(50)	0.52(0.51)	0.54(0.53)	0.49(0.51)	0.52(0.52)
	Stanford NER	0.82(0.8)	0.40(0.48)	0.53(0.58)	**0.62**(0.61)	0.34(0.35)	0.44(0.45)
	OpenCalais	0.46(0.45)	0.43(0.37)	0.44(0.38)	0.45(0.44)	0.41(0.27)	0.43(0.33)

coverage on candidate extraction, the Z-score threshold for candidate classification is adjusted to a lower value. But the lack of candidate surface form repetition in WNUT17 and BTC can often result in noisy infrequent false positives, leading to TwiCS's low precision for this dataset. However, TwiCS still obtains a competitive performance in F1 measure due to its generally higher recall. This is rendered possible with the combination of syntax agnostic mention detection of Phase II and the classifier. Please note that unlike TwiCS, Aguilar et al. also used two sets of external resources – a Word2Vec embedding set prepared by Godin et al. [47] on Twitter and gazetteers constructed by [17] – and POS-tags generated using [48].

For effectiveness study on real Twitter streams that conform to the E-H1 hypothesis, we use a total of three test sets (D_1 through D_3) varying both in size and topic coverage. To the best of our knowledge, this expands the scope of our effectiveness evaluation in size (over 10K tweets in total) than that conducted by baselines (like TwitterNLP) or the third party datasets. We also conduct the evaluation in a staggered fashion by separately evaluating datasets of different sizes. This helps us better gauge the impact of size and topic diversity of dataset on the performance of TwiCS. D_1 is the smallest dataset with 1,000 tweets from 1 topic. D_2 is generated by extracting 3K tweets (≈5K sentences) seeded with three topics consisting of six different top-trending hashtags (2 hashtags/topic). D_3 is the largest with over 6,000 tweets from 5 topics.

Annotation of each tweet in these datasets has been independently conducted by two graduate students. Disagreements are resolved by either arriving at mutual consensus or involving the supervision of one of the authors. Further annotation details can be found in TwiCS' Github page.

For training, we extract sample entity candidates and their syntactic distributions with a separate dataset D_4 with tweets of a single topic. They are labelled and fed into the classifier described in Section 6.

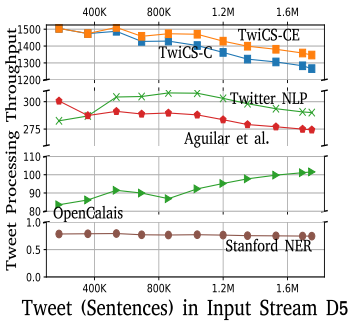
For efficiency study, we use two larger datasets (D_5 and D_6). The first consists of 1.035M tweets collected from 3 topics. It has a roughly uniform distribution of tweets per topic to avoid bias. D_6 is the 2011 NIST Tweets Collection (available at https://trec.nist.gov/data/tweets) and was part of the TREC 2011 Microblog track [49]. Of the original 16M tweet IDs, some are no longer accessible due to the tweet or its user account being taken down. The NIST provided twitter-tools API can recover 10.26M tweets that were fed to TwiCS as a continuous message stream to test its EMD efficiency on large streams. Note that unlike our other datasets, the tweets in D_6 are not topically coupled from selective hashtags. They are meant to be a representative sample of the Twittersphere, during the course of sixteen days when it was collected. This further tests TwiCS' performance by removing the scope of relying on hypothesis E-H1.

Performance Metrics: In compliance with WNUT17, we compare TwiCS' performance with all baseline systems for two tasks. We use Precision (P), Recall (R) and F1-score to evaluate effectiveness for these tasks. The first task, **EMD**, requires the identification of all occurrences of entities in their various surface forms within the dataset and is captured in WNUT17 as **F1 (Surface)**. **ED** requires the discovery of unique entities in a dataset and is measured as **F1 (entity)** in WNUT17. Note that since TwiCS currently focuses only on detecting entity mentions, our evaluation excludes type classification when considering performance for either task.

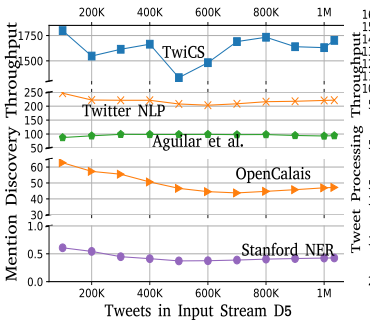
8.1 Effectiveness

Table 4 gives the outcome of our effectiveness analyses on D_1,D_2 and D_3 for EMD and ED. We report the average performance using 5-fold cross-validation with 4 folds to tune the Z-score threshold and one fold as the test data.

TwiCS outperforms the state-of-the-art for both ED and EMD tasks (see the numbers in Table 4 that are not within parentheses). For EMD, TwiCS's F1-score is about 17.2% better than the best baseline, Aguilar et al. on D_1. The improvement over the best state-of-the-art F1-score on D_2 and D_3 are 12.8% and 13.8%, respectively. Although BERTweet is the latest state-of-the-art in NER, its performance lags behind both TwiCS and Aguilar et al. Performance gains for TwiCS are mostly from better Recall, while still achieving good precision. The high quality candidate extraction of TwiCS owing to good direct coverage, followed by its ability to recognize different mention variations of the same entity due to high indirect coverage in streams, is the main

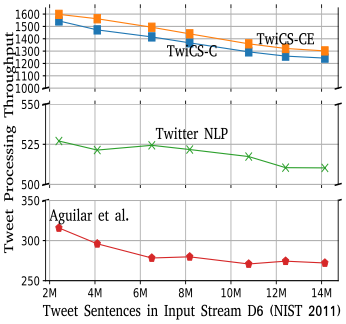


(a) Tweet Processing

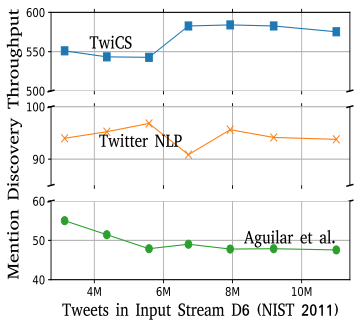


(b) Mention Discovery

Fig. 8: Throughput of TwiCS vs. state-of-the-art on D5



(a) Tweet Processing



(b) Mention Discovery

Fig. 9: Throughput of TwiCS vs. state-of-the-art on D6

reason why it performs so much better for EMD. TwiCS is able to eliminate many one-pass false negatives using boundary segmentation and classification in later cycles. Since ED concerns with only detecting unique entity strings in the dataset, irrespective of all its mentions, the efficacy of TwiCS is slightly reduced for this task. Nonetheless, it still demonstrates superior performance compared to the baselines. For ED, the improvement in TwiCS's F1-score is nearly 17.8% on D_1, 3.2% on D_2, and 1.6% on D_3, over that of the best baseline numbers.

Since Stanford NER is pre-conditioned to detect entities of only limited types, we also measure effectiveness with a set of alternate annotations, limited to types covered by all methods (Person, Location and Organization). This ensures a fairer comparison across all systems. The results for this evaluation are also provided in Table 4 (see numbers within parentheses). Note that the precision drops slightly for TwiCS, Aguilar et al. and Twitter NLP due to a number of previously legitimate EMs (of alternate types) now declared False Positives. However these are only a few in number, with majority of EMs belonging to these 3 common types.

TwiCS also performs better because it handles cases like:

(1) Incomplete Entity Mentions: This case is particularly frequent in Twitter where EMs tend to be more noisy than in a formal setting. For example,*'Trump'* as a candidate was consistently more frequent than *'Donald Trump'*. Systems like OpenCalais do not handle Incomplete (or Partial) Mentions particularly well, e.g., it was unable to identify *'Trump'* as an entity. At this point, we do not make any distinction in the treatment of partial and complete mentions for the ED task. For example, *'Trump'* and *'Donald Trump'* are treated as distinct and valid entity candidates, if identified as such. We leave the clustering of partial mentions with complete entity mentions into a unified canonical form as a future work.

(2) Multiple Mention Variations of the Same Entity: The baselines exhibit subtle differences in performance of the two tasks. In Table 5, we present the variation in output of Stanford NER for different Mentions of the same entity. TwiCS, on the other hand, can correctly extract all mention variations of an entity once it is discovered.

(3) Alphanumeric Entities and Abbreviations: Entities like Maroon 5/Win7 and abbreviations are fairly frequent in tweets. We do not make any distinction in their handling.

A **weakness** of using frequency-based method for ED is that less frequent entities are unlikely to be recommended as candidates for classification, impacting the recall. We study this later in Section 8.3. This is consistent with the observed superior performance of Aguilar et al. and BERTweet on identifying less frequent true positives, which TwiCS does not find. Supervised techniques can overcome this impediment as they do not rely on frequency estimates, and consider immediate semantic context to recommend candidates. Aguilar et al. also uses external word embeddings and gazetteers that are particularly useful in this regard.

8.2 Efficiency

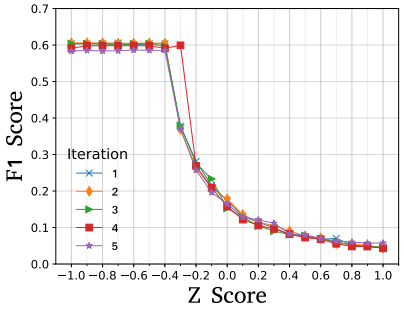
The two larger datasets D_5 and D_6 are used to measure efficiency. For these test sets, we execute TwiCS over multiple batches to mimic its behaviour in an online setting, with continuously incoming batches of tweets. The batch size is set to 100K tweets for the execution of both datasets.

The only other online system in our baselines is OpenCalais, but its offline version is not yet available for third-party usage. OpenCalais requires its client to send out input in batches, but the batch-size needs to be typically low (set to 1,000 tweets per batch) to avoid receiving an HTTP Error Code 500. Being a web service, OpenCalais has to balance its access by hundreds of clients. Since it does not explicitly return timing estimates with its output packets, we had to determine efficiency by measuring the response time for individual packets. However, this might include the response latency caused by OpenCalais' internal load-balancing mechanism and might not be the accurate indicator of the efficiency of the real system behind. On the other hand, systems like OpenCalais often operate on more powerful servers than non-commercial systems. Therefore, we note that an exact efficiency comparison with OpenCalais is difficult to conduct.

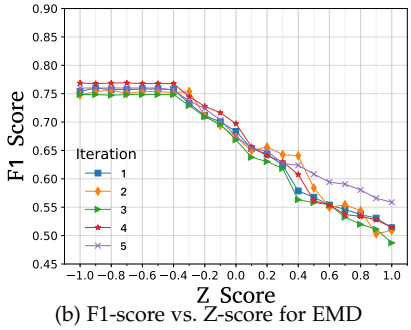
Both Stanford NER and OpenCalais take near-constant time per tweet sentence, but are significantly slower than the rest. Executing these systems on D_6 therefore required a very long time and was difficult to finish in a timely manner. Additionally, continuous iterative execution of D_6 on OpenCalais, like a large message stream, resulted in sending too many API calls and consequent unsuccessful responses. TwiNER also yielded a constant low throughput by processing 0.06 tweet-sentences per second. For these

TABLE 5: Detecting multiple Mention Variations

TweetText	Stanford	TwiCS
spicer should resign	-	spicer
Spicer should resign	Spicer	Spicer
we demand spicer's resignation	-	spicer



(a) F1-score vs. Z-score for ED



(b) F1-score vs. Z-score for EMD

Fig. 10: Change in performance with Z-score threshold

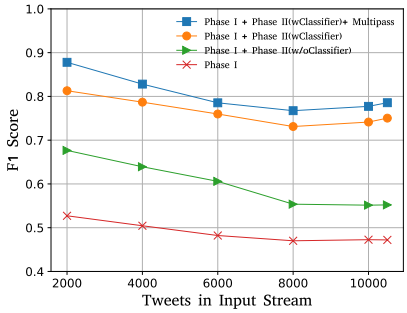


Fig. 11: Impact on Performance of Components

reasons, we present the efficiency results of only TwiCS, Gaguilar at al. and Twitter NLP on D_6. We compare the efficiency of these systems using two types of throughput:

(1) **Tweet Processing Throughput**: Existing baselines all employ a once-and-done approach when processing tweets. In this case, throughput is simply the number of tweet sentences processed per time unit (second). In contrast, a tweet-sentence in TwiCS remains incomplete until all of its EM candidates are confidently labelled, i.e., tweet sentences may be processed multiple times before completion or eviction. To this effect, we define two measures of throughput for TwiCS. The first is the Tweet Completion Throughput – the number of tweet sentences marked completely processed by TwiCS (curve TwiCS-C in Figure 8a and Figure 9a) per second. The second is the Tweet Processing Throughput – the number of sentences exiting the system per second, either completely processed or evicted (curve TwiCS-CE in Figure 8a and Figure 9a). Incomplete sentences, accumulating at any given processing cycle, do not contribute towards either of these measures. This explains the fluctuation in TwiCS's throughputs over time compared to other baselines.

(2) **Mention Discovery Throughput**: Measures the number of EMs discovered in the stream per time unit. On average, TwiCS discovers approximately 1603 mentions per second on D_5 and 566 mentions per second on D_6.

Figures 8 and 9 show the performance of TwiCS and other baselines on D_5 and D_6, with respect to the above two throughput metrics. TwiCS consistently outperforms all the baselines. Its average Tweet Completion Throughput is approximately 3.67 times of the throughput of Twitter NLP on D_5. For D_6, the average completion throughput is 2.75 times of that of Twitter NLP. The Tweet Processing throughput, in both cases, is higher than the corresponding Tweet Completion throughputs, as is evident from the figures. Despite the overhead caused by its multi-pass framework, the gain in TwiCS's efficiency is due to the computationally inexpensive nature of individual phases.

Additionally, note that the Mention Discovery Throughput on a dataset is dependent on its mention density, i.e., the number of entity mentions per tweet sentence of the dataset. A large yet mention-sparse dataset will result in low Mention Discovery Throughput. This is consistent with the low Mention Discovery Throughput of all system on D_6, which is a vast collection randomly sampled from Twitterverse and includes spam tweets. However, even for D_6, the improvement of TwiCS's Mention Discovery Throughput over other systems persist. The throughput here is approximately 6 times of Twitter NLP and 10.4 times of Aguilar et al.

8.3 Independent Analysis of Our Approach

We now analyze the impact of individual components and parameters of TwiCS through a number of experiments.

1. Z-score threshold: As mentioned before, we conduct 5-fold cross-validation for all our annotated datasets and tune the **Z-score threshold** parameter to its optimal value. Each iteration in Figure 10, corresponds to one of five train-test splits. The optimal value for a test fold is the one which maximizes effectiveness on its training fold. Performance is evaluated in 2 dimensions: 1) Entity Discovery over CandidateMentionBase, and 2) Tweet-level Mention Detection. In our experiment, the F1-score was maximized at a Z-score of -0.7, across all test partitions.

2. Contribution of Individual Components: For evaluating the contribution of the different elements of our multi-pass framework towards TwiCS's EMD effectiveness, we use the entire collection of manually annotated tweets as the test set. We have 4 variants of TwiCS. Figure 11 shows the improvement in performance as individual system components are added. From bottom to top, the first curve (with only Phase I) reports the weakest performance proving just a CS guided candidate identification is not enough. The third curve from bottom is TwiCS' once-and-done version while the topmost one is multi-pass enabled TwiCS. The performance gain makes it clear why systems like TwiCS that prioritized computational efficiency over sophistication, surely benefits by having a selectively multi-pass strategy in place. We will also establish shortly the fact that the multi-pass overhead of TwiCS is fairly minute, considering the improvement achieved.

3. Limitations: Here we analyze two main factors that affect the performance of TwiCS.

(1) **Memory consumption**: The multi-pass approach with a simple eviction strategy (last paragraph of Section 7) retains all incomplete tweet sentence until their eviction threshold is reached. This will incur additional memory consumption. We track the additional memory overhead incurred due to retention of incomplete sentences during the execution of D_5 in Figure 12. Compared to the incoming batch size (which is the only input retention for traditional once-and-done EMD executions), the multi-pass overhead (with additional incomplete sentences) is minimal – approximately 5.11% of the once-and-done counterpart.

For D_6, the total size for 10.26M tweets is approximately 1.3GB. With a batch size of 100K, the average memory use for multipass TwiCS was 113MB per batch with a similar 5.09% overhead. We plan to investigate more efficient strategies for incomplete tweets to further limit the overhead.

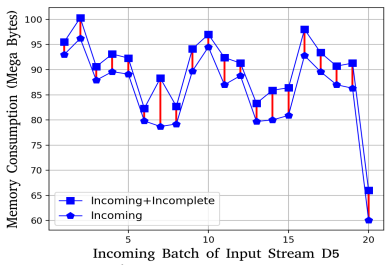


Fig. 12: Multi-pass Memory Consumption

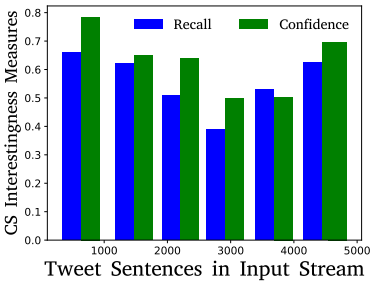


Fig. 13: CS-guided Candidate Extraction

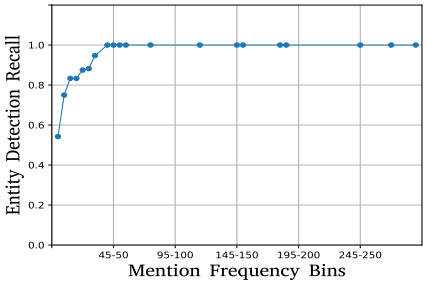


Fig. 14: Impact of Frequency on ED Recall

(2) **CS-guided Candidate Extraction:** The histograms in Figure 13 depict the outcome of our evaluation of the CS Heuristic on our three annotated datasets, using interestingness measures common in information retrieval. The confidence estimation denotes the likelihood of a CS occurrence being an EM. Recall denotes the percentage of actual EMs captured by the CS Heuristic.

A limitation of TwiCS is that it can only recognize an entity and its mentions if the entity appears as a CS at least once in the tweet stream. Let's call entities that appear as a CS at least once in a dataset as *CS entities* and other entities as *non-CS entities*. Our analysis of datasets (D_1 to D_3) shows that CS entities appear significantly more times than non-CS entities on average. This implies (1) more important entities and EMs are more likely to be discovered by TwiCS and (2) the discovered entities contribute more to EM discovery as they correspond to more EMs. Of the 283 entities in D_1, 211 (74.5%) are CS entities and 72 (25.4%) are non-CS entities. Additionally, on average, each CS entity appears 5.65 times (i.e., has 5.65 mentions) and each non-CS entity appears 2.3 times. As a result, the 211 CS entities contribute to nearly 88% EMs while the 72 non-CS entities deliver only about 12% EMs. In D_2, there are 906 unique entities, among which 331, or about 36.5%, are non-CS entities and the remaining are CS entities. The non-CS entities appear a total of 405 times – averaging a mere 1.22 times per entity. In contrast, the 575 CS entities appear a total of 2,966 times – averaging more than 5 times each and contributing to 88% of the EMs. Dataset D_3 also exhibits a similar behavior. There are 675 unique entities among which 225 (33%) are non-CS entities and they only contribute 539 (about 9%) of the EMs in D_3 – averaging at 2.39 times per entity – as opposed to the CS entities that generate 5,554 mentions, averaging at 12.34 times and contributing to more than 91% of the total EMs.

From these numbers, it is reasonable to speculate that, in general, when an entity appears a sufficient number of times (5 or above for our datasets), it is likely to appear as a CS at least once. In larger datasets (the annotated datasets used in our experiments are rather small), higher percentage of entities would likely become CS entities. As TwiCS is designed to target large datasets, it is expected to perform well for real-world message streams. Finally, despite this limitation, TwiCS still significantly outperforms the baseline state-of-the-art methods in stream settings.

Figure 14 illustrates how well TwiCS is able to detect high-frequency entities in datasets D_1-D_3. We group entities of different mention frequency in bins of width 5 and track the Entity Detection Recall that TwiCS yields for each bin. For low frequency entities the recall is lower, indicating more mislabeling. But TwiCS is able to consistently discover high frequency entities. This validates the intuition that TwiCS is able to learn better the more mentions it sees of an entity.

9 Conclusions

In this paper, we presented the design and evaluation of a novel system, TwiCS, for EMD/ED from microblog streams. The approach behind this system is guided by several empirically validated principles about entity mentions in microblog streams. TwiCS is unique in that it consists of two novel phases, where the first phase performs a linguistically shallow and computationally light analysis of incoming tweets to quickly detect entity candidates while the second phase extracts more mentions of said candidates and validates true EMs among them. Furthermore, as a departure from once-and-done solutions, the second phase strategically reprocesses a small percentage of tweets. Based on this approach, TwiCS processes tweet streams over multiple iterations with each iteration first focusing on refining a set of seed entity candidates and then verifying entities from them to detect entity mentions. We conducted extensive experiments using a variety of datasets to show that for EMD/ED from microblog streams TwiCS is able to achieve significantly higher effectiveness as well as much higher efficiency compared with several state-of-the-art systems. A key insight to TwiCS's good performance is that for tweets with entity mentions that cannot be identified confidently at the first attempt, additional passes can recover initially missed mentions, which is particularly effective for improving recall. We believe that this insight is also applicable to ED and EMD in many non-English languages. For future work, we aim to grow a complete IE pipeline that performs Entity Detection and Linkage on tweet streams at scale.

Acknowledgements.. This work was supported in part by the U.S. National Science Foundation BIGDATA 1546480 and 1546441 grants.

References

[1] L. Derczynski, E. Nichols, M. van Erp, and N. Limsopatham, "Results of the wnut2017 shared task on novel and emerging entity recognition," in *WNUT*, 2017, pp. 140–147.

[2] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by gibbs sampling," in *ACL*, 2005, pp. 363–370.

[3] A. McCallum and W. Li, "Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons," in *CONLL*, 2003, pp. 188–191.

[4] A. Ritter, S. Clark, Mausam, and O. Etzioni, "Named entity recognition in tweets: An experimental study," in *EMNLP*, 2011, pp. 1524–1534.

[5] F. Dernoncourt, J. Y. Lee, and P. Szolovits, "NeuroNER: an easy-to-use program for named-entity recognition based on neural networks," 2017.

[6] G. Aguilar, S. Maharjan, A. P. López-Monroy, and T. Solorio, "A multi-task approach for named entity recognition in social media data," in *W-NUT*. ACL, 2017.

[7] M. N. Gerguis, C. Salama, and M. W. El-Kharashi, "Asu: An experimental study on applying deep learning in twitter named entity recognition," *WNUT*, 2016.

[8] N. Reimers and I. Gurevych, "Optimal hyperparameters for deep lstm-networks for sequence labeling tasks," *arXiv:1707.06799*, 2017.

[9] K. Bontcheva, L. Derczynski, A. Funk, M. A. Greenwood, D. Maynard, and N. Aswani, "TwitIE: An open-source information extraction pipeline for microblog text," in *RANLP*, 2013.

[10] X. Liu, S. Zhang, F. Wei, and M. Zhou, "Recognizing named entities in tweets," in *HLT*, 2011.

[11] C. Cherry and H. Guo, "The unreasonable effectiveness of word representations for twitter named entity recognition," in *HLT 2015*.

[12] E.-S. Yang and Y.-S. Kim, "Hallym: Named entity recognition on twitter with induced word representation," in *ACL-IJCNLP 2015*.

[13] Z. Toh, B. Chen, and J. Su, "Improving twitter named entity recognition using word representations," in *WNUT 2015*.

[14] E.-S. Yang, Y.-B. Kim, R. Sarikaya, and Y.-S. Kim, "Drop-out conditional random fields for twitter with huge mined gazetteer." in *HLT-NAACL*, 2016, pp. 282–288.

[15] M. S. Akhtar, U. K. Sikdar, and A. Ekbal, "Iitp: Multiobjective differential evolution based twitter named entity recognition," in *W-NUT*, 2015, pp. 61–67.

[16] T. Tian, M. Dinarelli, and I. Tellier, "Lattice: Data adaptation for named entity recognition on tweets with features-rich crf," in *ACL 2015 workshop WNUT*, 2015.

[17] S. Mishra and J. Diesner, "Semi-supervised named entity recognition in noisy-text," in *WNUT*, 2016, pp. 203–212.

[18] P. von Däniken and M. Cieliebak, "Transfer learning and sentence level features for named entity recognition on tweets," in *WNUT 2017*.

[19] F. Dugas and E. Nichols, "Deepnnner: Applying blstm-cnns and extended lexicons to named entity recognition in tweets," in *WNUT*, 2016.

[20] Q. Zhang, J. Fu, X. Liu, and X. Huang, "Adaptive co-attention network for named entity recognition in tweets." in *AAAI 2018*.

[21] Y. Shen, H. Yun, Z. C. Lipton, Y. Kronrod, and A. Anandkumar, "Deep active learning for named entity recognition," *arXiv preprint arXiv:1707.05928*, 2017.

[22] B. Strauss, B. Toma, A. Ritter, M.-C. de Marneffe, and W. Xu, "Results of the wnut16 named entity recognition shared task," in *WNUT 2016*.

[23] N. Limsopatham and N. H. Collier, "Bidirectional lstm for named entity recognition in twitter messages," in *COLING 2016*.

[24] J. Williams and G. Santia, "Context-sensitive recognition for emerging and rare entities," in *W-NUT*, 2017, pp. 172–176.

[25] P. Jansson and S. Liu, "Distributed representation, LDA topic modelling and deep learning for emerging named entity recognition from social media," in *W-NUT*, 2017, pp. 154–159.

[26] B. Y. Lin, F. Xu, Z. Luo, and K. Zhu, "Multi-channel bilstm-crf model for emerging named entity recognition in social media," in *WNUT 2017*.

[27] D. Q. Nguyen, T. Vu, and A. T. Nguyen, "BERTweet: A pre-trained language model for English Tweets," in *EMNLP*, 2020, pp. 9–14.

[28] C. Li, J. Weng, Q. He, Y. Yao, A. Datta, A. Sun, and B.-S. Lee, "Twiner: Named entity recognition in targeted twitter stream," in *SIGIR*, 2012, pp. 721–730.

[29] K. Wang, C. Thrasher, E. Viegas, X. Li, and B.-j. P. Hsu, "An overview of microsoft web n-gram corpus and applications," in *NAACL HLT 2010 DEMO*, 2010, pp. 45–48.

[30] X. Liu, M. Zhou, F. Wei, Z. Fu, and X. Zhou, "Joint inference of named entity recognition and normalization for tweets," in *ACL*, 2012.

[31] I. Yamada, H. Takeda, and Y. Takefuji, "Enhancing named entity recognition in twitter messages using entity linking," in *WNUT 2015*.

[32] T. Reuters, "Opencalais," 2008. [Online]. Available: www.opencalais.com

[33] A. Gattani, D. S. Lamba, N. Garera, M. Tiwari, X. Chai, S. Das, S. Subramaniam, A. Rajaraman, V. Harinarayan, and A. Doan, "Entity extraction, linking, classification, and tagging for social media: a wikipedia-based approach," *PVLDB*, vol. 6, no. 11, pp. 1126–1137, 2013.

[34] B. Y. Lin and W. Lu, "Neural adaptation layers for cross-domain named entity recognition," *arXiv preprint arXiv:1810.06368*, 2018.

[35] N. Peng and M. Dredze, "Multi-task domain adaptation for sequence tagging," *arXiv preprint arXiv:1608.02689*, 2016.

[36] S. Ashwini and J. D. Choi, "Targetable named entity recognition in social media," *arXiv preprint arXiv:1408.0782*, 2014.

[37] M. Bernstein, B. Suh, L. Hong, J. Chen, S. Kairam, and E. H. Chi, "Eddi: Interactive topic-based browsing of social status streams," in *UIST*, 2010.

[38] W. Xie, F. Zhu, J. Jiang, E.-P. Lim, and K. Wang, "Topicsketch: Real-time bursty topic detection from twitter," in *TKDE*, 2016.

[39] W. Hua, K. Zheng, and X. Zhou, "Microblog entity linking with social temporal context," in *SIGMOD*, 2015.

[40] W. Contributors, "Capitalization — Wikipedia, the free encyclopedia." [Online]. Available: "https://en.wikipedia.org/wiki/Capitalization"

[41] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *JMLR*, vol. 9, no. Aug, pp. 1871–1874, 2008.

[42] K. Church, W. Gale, P. Hanks, and D. Hindle, "Using statistics in lexical analysis," 1991.

[43] M. Jiang, J. Shang, T. Cassidy, X. Ren, L. M. Kaplan, T. P. Hanratty, and J. Han, "Metapad: Meta pattern discovery from massive text corpora," in *SIGKDD*, 2017, pp. 877–886.

[44] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, pp. 2579–2605, 2008.

[45] K. Lee, D. Palsetia, R. Narayanan, M. M. A. Patwary, A. Agrawal, and A. Choudhary, "Twitter trending topic classification," in *ICDM*, 2011.

[46] L. Derczynski, K. Bontcheva, and I. Roberts, "Broad Twitter corpus: A diverse named entity recognition resource," in *COLING*, 2016, pp. 1169–1179.

[47] F. Godin, B. Vandersmissen, W. De Neve, and R. Van de Walle, "Multimedia lab@ acl wnut ner shared task: Named entity recognition for twitter microposts using distributed word representations," in *W-NUT*, 2015, pp. 146–153.

[48] L. Kong, N. Schneider, S. Swayamdipta, A. Bhatia, C. Dyer, and N. A. Smith, "A dependency parser for tweets," 2014.

[49] I. Soboroff, I. Ounis, C. Macdonald, and J. J. Lin, "Overview of the trec-2012 microblog track." in *TREC*, vol. 2012, 2012.

PLACE PHOTO HERE

Satadisha Saha Bhowmick is currently pursuing her PhD in Computer Science from Binghamton University. Her research interests include Information Extraction and Information Retrieval.

PLACE PHOTO HERE

Eduard C. Dragut received the PhD degree in computer science from the University of Illinois at Chicago in 2010. He is currently an associate professor in the Department of Computer and Information Sciences, Temple University. His research interests include web-based information retrieval, cleaning, integration, and mining. He is a coauthor of the book *Deep Web Query Interface Understanding and Integration*. He co-chaired multiple workshops, the last of which are DaSH@KDD and DaSH@NAACL in 2021. He is a member of the IEEE.

PLACE PHOTO HERE

Weiyi Meng received the BS degree in mathematics from Sichuan University, China, in 1982, and the MS and PhD degrees in computer science from the University of Illinois at Chicago, in 1988 and 1992, respectively. He is currently a professor in the Department of Computer Science at the State University of New York at Binghamton. His research interests include web-based information retrieval, metasearch engines, and web database integration. He is a coauthor of three books *Principles of Database Query Processing for Advanced Applications*, *Advanced Metasearch Engine Technology*, and *Deep Web Query Interface Understanding and Integration*. He has published more than 150 technical papers. He is a senior member of the IEEE.