Entity Extraction, Linking, Classification, and Tagging for Social Media: A Wikipedia-Based Approach

Abhishek Gattani[1], Digvijay S. Lamba[1], Nikesh Garera[1], Mitul Tiwari[2], Xiaoyong Chai[1], Sanjib Das[3], Sri Subramaniam[1], Anand Rajaraman[4], Venky Harinarayan[4], AnHai Doan[1,3]

[1]@WalmartLabs, [2]LinkedIn, [3]University of Wisconsin-Madison, [4]Cambrian Ventures

ABSTRACT

Many applications that process social data, such as tweets, must extract entities from tweets (e.g., "Obama" and "Hawaii" in "Obama went to Hawaii"), link them to entities in a knowledge base (e.g., Wikipedia), classify tweets into a set of predefined topics, and assign descriptive tags to tweets. Few solutions exist today to solve these problems for social data, and they are limited in important ways. Further, even though several industrial systems such as OpenCalais have been deployed to solve these problems for text data, little if any has been published about them, and it is unclear if any of the systems has been tailored for social media.

In this paper we describe in depth an end-to-end industrial system that solves these problems for social data. The system has been developed and used heavily in the past three years, first at Kosmix, a startup, and later at WalmartLabs. We show how our system uses a Wikipedia-based global "real-time" knowledge base that is well suited for social data, how we interleave the tasks in a synergistic fashion, how we generate and use contexts and social signals to improve task accuracy, and how we scale the system to the entire Twitter firehose. We describe experiments that show that our system outperforms current approaches. Finally we describe applications of the system at Kosmix and WalmartLabs, and lessons learned.

1. INTRODUCTION

Social media refers to user generated data such as tweets, Facebook updates, blogs, and Foursquare checkins. Such data has now become pervasive, and has motivated numerous applications in e-commerce, entertainment, government, health care, and e-science, among others.

Many such applications need to perform entity extraction, linking, classification, and tagging over social data. For example, given a tweet such as "Obama gave an immigration speech while on vacation in Hawaii", *entity extraction* determines that string "Obama" is a person name, and that "Hawaii" is a location. *Entity linking* goes one step further, inferring that "Obama" actually refers to an entity in a knowledge base, for example, the entity at URL en.wikipedia.org/wiki/Barack_Obama, and that "Hawaii" refers to the entity at URL en.wikipedia.org/wiki/Hawaii. *Classification* assigns a set of predefined topics to the tweet, such as "politics" and "travel". Finally, *tagging* assigns descriptive tags to the tweet, such as "politics", "tourism", "vacation", "President Obama", "immigration", and "Hawaii", the way a person may tag a tweet today.

Entity extraction, a.k.a. named entity recognition (NER), and text classification are well-known problems that have been around for decades (e.g., [4, 23]), while entity linking and tweet tagging are newer problems that emerged in the past few years [27]. Nevertheless, because of their importance to a large variety of text-centric applications, these problems have received significant and increasing attention.

Despite this attention, few solutions exist today to solve these problems for social media, and these solutions are limited in several important ways. First, the solutions often "recycle" techniques developed for well-formed English texts. A significant amount of social data, however, are misspelled ungrammatical short sentence fragments, thereby proving ill-suited for these techniques. Second, the solutions often employ computation-intensive techniques that do not scale to high-speed tweet streams of 3000-6000 tweets per second.

Third, existing solutions typically do not exploit context information, such as topics that a Twitter user often tweets about. As we show in this paper, since many types of social data (especially tweets and Facebook updates) are often very short (e.g., "go Giants!"), it is critical that we infer and exploit context information to improve accuracy. Fourth, existing solutions typically do not exploit social signals, such as traffic on social sites (e.g., Wikipedia, Pinterest), even though such signals can greatly improve accuracy.

Finally, most current solutions address only a single problem, in isolation, even though as we show later in this paper, addressing all four problems in a synergistic fashion can further improve the overall performance.

In the past few years, several industrial systems to extract, link, classify and tag text data, such as OpenCalais at opencalais.com, have also been deployed on the Web (see the related work section). However, little, if any, has been published about these systems, and as far as we know, none of these deployed systems has been specifically tailored for social media.

In this paper we describe an end-to-end industrial system that extracts, links, classifies, and tags social data. To the best of our knowledge, this is the first paper that de-

Proceedings of the VLDB Endowment, Vol. 6, No. 11

scribes such a system in depth. The system has been developed and used heavily since 2010, first at Kosmix, a startup that performed semantic analysis of social data, then later, since mid-2011 at WalmartLabs, a research and development lab for Walmart (which acquired Kosmix). At WalmartLabs, the system has been used extensively to process tweets, Facebook updates, and other types of social data, to power a variety of e-commerce applications (see Section 5).

Even though our system can handle many types of social data (as well as a variety of text documents, see Section 5), for expository reasons in this paper we will focus on handling tweets. Our system differs from current systems in the following important ways:

Using a Global and "Real-Time" Knowledge Base: Our knowledge base (which we use to find and link to entities mentioned in tweets) is built from Wikipedia (see [13]). Wikipedia is global, in that it contains most concepts and instances judged important in the world. Thus, it provides a good coverage for the tasks. More importantly, it is "real time" in that contributors continuously update it with new entities that just appear in real-world events. This "real time" nature makes it especially well-suited for processing social data, and in fact, we take certain measures to make it even more "real time" (see Section 3.1). In contrast, many current solutions use knowledge bases that are updated less frequently.

Synergistic Combination of the Tasks: Our system interleaves the four tasks of extraction, linking, classification, and tagging in a synergistic fashion. For example, given a tweet, we begin by performing a preliminary extraction and linking of entity mentions in that tweet. Suppose many such mentions link to many nodes under the subtree "Technology" in our knowledge base (KB). Then we can infer that "Technology" is a likely topic for the tweet, thereby helping classification. In return, if we have determined that "Technology" is indeed a topic for the tweet, then we can infer that string "apple" in the tweet likely refers to the node "Apple Corp." in the KB, not the node "Apple (fruit)", thereby helping entity linking.

Using Contexts and Social Information: Given a tweet such as "go Giants!", without some contexts, such as knowing that this user often tweets about the New York Giants football team, it is virtually impossible to extract and link entities accurately. As another example, it is not possible to process the tweet "mel crashed, maserati gone" in isolation: we have no idea which person named Mel the user is referring to. However, if we know that in the past one hour, when people tweeted about Mel Gibson, they often mentioned the words "crash" and "maserati" (a car brand), then we can infer that "mel" likely refers to the node Mel Gibson in the KB. Our system exploits such intuitions. It collects contexts for tweets, Twitter users, hash tags, Web domains, and nodes in the KB. It also collects a large number of social signals (e.g., traffic on Wikipedia and Pinterest pages). The system uses these contexts and signals to improve the accuracy of the tasks.

Other important features of our system include a minimal use of complex time-intensive techniques, to ensure that we can process tweets in real time (at the rate of up to 6000 tweets per second), and the use of hand-crafted rules at various places in the processing pipeline to exert fine-grained control and improve system accuracy.

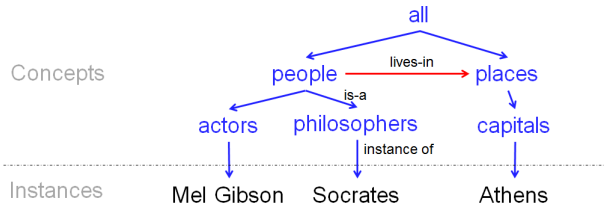

Figure 1: A tiny example of a KB

In the rest of this paper we first define the problems of entity extraction and linking, and tweet classification and tagging. We then describe the end-to-end system in detail. Next, we present experiments that show that the current system outperforms existing approaches. Next, we briefly describe several e-commerce and consumer-facing applications developed at Kosmix and WalmartLabs that make use of this system, and discuss lessons learned. We conclude with related work and a discussion of future work.

2. PROBLEM DEFINITION

We now describe the problems considered in this paper. As mentioned in the introduction, here we focus on processing tweets (see Section 5 for examples of other kinds of data that we have applied our system to).

Tweets and Tweet Stream: For our purpose, a tweet has a user ID and a text. For example, the user with the Twitter ID @polwatcher tweets the text "Obama just left for Hawaii". (Tweets often have far more data, such as time and location, but we do not consider them in this paper.) Many Kosmix and WalmartLabs applications must process the entire Twitter firehose (i.e., a stream that emits 3000-6000 tweets per second) in real time. So an important requirement for our solution is that it scales to the firehose stream, i.e., can process tweets as fast as they come in.

Knowledge Base: As discussed earlier, we use a large knowledge base in our solution. A knowledge base (KB) typically consists of a set of concepts $C_1, \ldots, C_n$, a set of instances I_i for each concept C_i, and a set of relationships $R_1, \ldots, R_m$ among the concepts.

We distinguish a special relationship called "is-a", which specifies that a concept A is a kind of a concept B (e.g., Professors is a kind of People). The "is-a" relationships imposes a taxonomy over the concepts C_i. This taxonomy is a tree, where nodes denote the concepts and edges the "is-a" relationships, such that an edge $A \rightarrow B$ means that concept B is a kind of concept A. Figure 1 shows a tiny KB, which illustrates the above notions.

In many KBs, if A is a parent node of B and C (and only of these nodes) in the taxonomy, then the set of instances of A is the union of the instances of B and C. In our context, we do not impose this restriction. So node A may have instances that do not belong to B or C. (KBs typically also contain many domain integrity constraints, but we will not discuss them in this paper.)

In Section 3.1 we briefly discuss how we build our KB out of Wikipedia, then enrich it with a variety of structured data sources (e.g., MusicBrainz, City DB, Yahoo! Stocks, Chrome, Adam).

Categories: To perform entity extraction, we have defined a large set of entity categories that many real-world applications care about. This set contains not only person, location, and organization – the three most common categories that existing entity extraction works have studied, but also other common but less-studied categories such as product, music album, song, book, TV show, sport event, car, and movie.

We are now in a position to define the problems of entity extraction, linking, classification, and tagging considered in this paper.

Entity Extraction: In this problem, given a tweet, we want to locate strings in the text of the tweet that refer to the predefined categories. For example, given "Obama just went to Hawaii", we want to infer that "Obama" is a person name, and that "Hawaii" is a location. Given "just saw salt tonite", we want to infer that "salt" refers to a movie. We refer to strings such as "Obama", "Hawaii", and "salt" as *entity mentions*, or *mentions* for short.

This is the same problem considered by prior work in entity extraction (a.k.a. named entity recognition). However, most such works have considered only a small set of categories, typically person, location, and organization. In contrast, here we consider a far larger set of categories.

Entity Linking: Given a tweet t and a KB, we want to find strings in t that mention concepts and instances in the KB, and link these strings to the concepts and instances. For example, given Wikipedia as a KB, we want to link "Obama" in tweet "Obama just went to Hawaii" to the person instance en.wikipedia.org/wiki/Barack_Obama, and "Hawaii" to the state instance en.wikipedia.org/wiki/Hawaii. We often refer to a pair of (entity mention, node) such as ("Obama", en.wikipedia.org/wiki/Barack_Obama) also as a *mention*, when there is no ambiguity.

The concepts and instances in the KB are often collectively referred to as *entities*, and the problem is often called *entity linking*. This problem is relatively new (emerged in the past decade), but is receiving increasing attention (see the related work section).

We note that entity linking is related, but different from entity extraction. For example, given the tweet "just saw salt tonite with charlie", entity extraction may infer that "salt" is a movie and that "charlie" is a person name. But it does not have to link these strings to any KB. In contrast, entity linking may merely infer that "salt" refers to a movie entity in the KB, if Salt indeed exists as a movie entity in that KB. It does not have to infer that "charlie" is a person name, if "charlie" refers to no entity in the KB.

Tweet Classification: In our setting, we have defined 23 topics, which correspond to 23 nodes that are the children of the root of the taxonomy in our KB. Example topics include politics, entertainment, technology, and travel. Given a tweet we want to classify it into one or several of these topics.

Tweet Tagging: Given a tweet, we want to tag it with descriptive tags, similar to the way a person may tag a tweet, or an author may tag a blog (e.g., to build a tag cloud later).

A key question is where these tags come from. In our setting, we consider the *names* of all concepts and instances in our KB to be possible tags. Given a tweet, our goal is to select from this universe of tags a small set of tags that best describe the tweet. For example, we may tag "Obama just gave an immigration speech in Hawaii" with Obama, Hawaii, US Politics, immigration, and vacation. This problem is also referred to as *social tagging*.

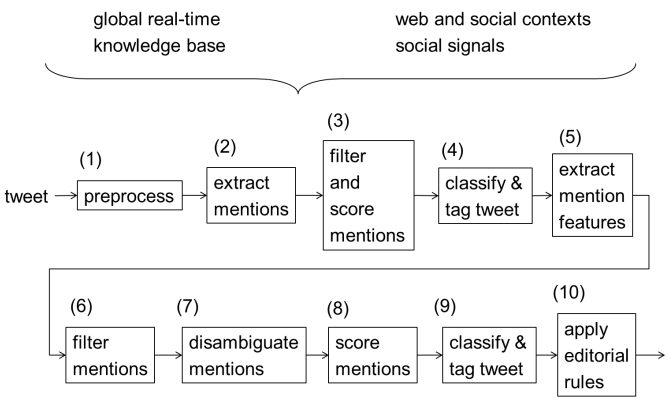

Figure 2: Our solution architecture

Practical Considerations: In our work, we quickly found that entity linking could become "excessive" given a large KB. For example, virtually *any* word in the tweet "I was happy to see you and Charlie together" can be linked to a page in Wikipedia (there are in fact several Wikipedia pages for "I" alone). This is clearly not necessary for many real-world applications. Thus, instead of finding *all* strings in the tweet that can be linked to entities in the KB, we try to find only strings that "best" describe the tweet, with "best" being a subjective notion that a particular application can control, using customized scoring functions and hand-crafted rules. For example, given "Obama just gave a speech in Hawaii", we extract and link "Obama" and "Hawaii", but not "just", "gave", "a", "speech", and "in".

3. SOLUTION ARCHITECTURE

We now describe the Kosmix/WalmartLabs solution to the above extraction, linking, classification, and tagging problems. Figure 2 shows the ten main steps of the solution. Given a tweet, we preprocess it, e.g., detecting the language, tokenizing (Step 1). Next, we use the KB to extract mentions from the tweet, remove certain mentions, then score the remaining ones (Steps 2-3). Here a mention refers to a pair of (string, KB node) such as ("Obama", en.wikipedia.org/wiki/Barack_Obama). So we are effectively performing entity extraction and linking at the same time. Then in the next step (Step 4) we use these mentions to classify and tag the tweet.

Next, we go back to processing the mentions, but do so in more depth. Specifically, we extract 30+ mention features, remove certain mentions using rules involving these features, disambiguate the mentions (e.g., linking "apple" to Apple the company not Apple the fruit), then score the mentions again (Steps 5-8). Next, we use the "clean" mentions to classify and tag the tweet again. Finally we apply hand-crafted editorial rules to filter mentions and classification and tagging results (Steps 9-10).

The above ten steps make use of a global "real-time" KB, Web and social contexts, social signals, and hand-crafted rules, as illustrated in the figure. We now describe these

steps in more details. But before that, we describe how we build the KB and create the Web and social contexts.

3.1 Building a Global "Real-Time" KB

We begin by considering what kind of KB we should build. We observe that Twitter is quite diverse, in that tweets can be about virtually *anything*. It follows that to process tweets with high recall, we should use a *global* KB, i.e., a KB that contains most concepts and instances deemed important in the world. Examples of such KBs are Wikipedia, Freebase [7], DBpedia [5], and YAGO [35].

Twitter is also "real-time", in that new events, topics, and entities are introduced into Twitter all the time, at (near) real-world speed. Thus, to process tweets in real time, we should also use a "real-time" KB, i.e., a KB that quickly incorporates new events, entities, and topics, soon after real-world happenings.

For these reasons, we decided to use Wikipedia, a global "real-time" KB being continuously updated by a large army of volunteers. If an important new event, topic, or entity appears in the real world, very soon it is mentioned in Wikipedia, often in a freshly constructed Wikipedia page. Thus, by continuously crawling Wikipedia, we can build a dynamic, fresh, and timely KB.

Wikipedia however is not a KB in the traditional sense. For example, it is not a taxonomy, but rather a giant directed cyclic graph in which a node often has multiple paths to the root, called *lineages*. For example, concept "Forced Suicide" is a child of "Ancient Greek Philosophers" and "5th Century BC Philosophers", which in turn are children of "Philosophers", which is a child of "ROOT". So "Forced Suicide" has two lineages: Force Suicide - Ancient Greek Philosophers - Philosophers - ROOT, and Force Suicide - 5th Century BC Philosophers - Philosophers - ROOT.

Thus, we converted Wikipedia into a KB. In particular, we converted its graph structure into taxonomy, by finding for each concept a single "main" lineage, called the *primary lineage*. At the same time we keep all the other lineages around, to avoid losing information. Later in Section 3.6 we show how all such lineages can be used to classify and tag tweets.

We describe the process of converting Wikipedia into a KB in a recent paper [13]. Since the resulting KB still did not have enough coverage for our applications, we added even more concepts and instances to the KB, by adding data from many structured sources, such as Chrome (an automobile source), Adam (health), MusicBrainz (albums), City DB, and Yahoo Stocks (see [13]).

Finally, we observed that our KB still was not sufficiently "real-time" in certain cases. For example, a new event X may not be introduced into Wikipedia (and thus made it into our KB) until 1-2 hours after the event has happened. During this time people already tweet about X, and not having X in our KB makes it difficult to process such tweets. To address this problem, we built an event detection system (to be described in an upcoming paper) that monitors Twitter to detect new interesting events. The moment a new event is detected, we clean and add it to our KB. While we can only handle hundreds of events each day, these tend to be the most popular events, and in many cases this solution helps us add the new events (and associated entities and topics) to our KB far sooner than waiting for them to appear in Wikipedia.

3.2 Generating Web and Social Contexts

As discussed in the introduction, having contexts help us better process tweets. We now describe how we generate these contexts.

Contexts for Tweets, Users, Hashtags, and Domains: In its basic form, a tweet is just 140 characters (and often far fewer than that, such as "go Giants!"). To process such short tweets, we try to obtain more context information. In particular, we focus on the followings:

- *Web context for tweets:* If a tweet mentions a URL, we retrieve the article on that Web page (if any), extract the title and a snippet (typically the first few lines) of the article, then associate (i.e., store) this title/snippet pair with the tweet. (We do not store the entire article for time and space reasons.) We call the title/snippet pair *the Web context* of the tweet, since it captures information on the Web related to the tweet.

- *Social context for users:* For each user ID in Twitter (e.g., @polwatcher), we define a social context that is time dependent. To compute the social context for user U at time t, we retrieve the last k tweets of U up to time t (where k is pre-specified), tag them using the system described in this paper, then union the tags and compute average scores. This produces a set of (tag, score) pairs that indicate what user U has often talked about in the last k tweets before time t. For example, if the social context at the current time for @polwatcher is {(politics, 0.8), (obama, 0.9), (coffee, 0.6), (madison, 0.4), (wisconsin, 0.2)}, then this user has often tweeted about these topics in recent days.

- *Social context for hashtags and Web domains:* Similarly, we define and compute social contexts for hashtags and Web domains. To compute the social context for a hashtag h at time t, we retrieve k tweets that mention h up to time t, tag them, then union the tags and compute average scores. If a tweet mentions a URL that comes from a Web domain (e.g., cnn.com, patch.com), then we compute a social context for that domain in an analogous fashion.

Efficiently Computing the Contexts: As each tweet comes in from the Twitter firehose, we compute its Web context on the fly, in real time. Next, we apply the system described in this paper to extract, link, classify, and tag the tweet. Then we pass the tweet together with its tags to a context computing system. This system uses the tags to compute the social contexts for the user ID, the hashtags, and the Web domains of the tweet, as described above. (It also uses the tags to compute a social context for each node in the KB, as described below.) Finally, we store these social contexts, so that the current system can use them in processing new incoming tweets.

Contexts for the Nodes in the KB: Similarly, we define and compute Web and social contexts for each node in the KB. To compute a Web context for a node N, we retrieve the articles associated with N (e.g., the Wikipedia page, if any), tag them, then union the tags and average the scores. To compute a social context for node N, we retrieve the last k tweets that mention N (i.e., containing entities that are

linked to node N by our current system), tag the tweets, then union the tags and average the scores.

We compute the Web contexts for the nodes in the KB in an offline process (and refresh these contexts regularly, once every few days). We compute the social contexts for the nodes using the same system that computes social contexts for users, hashtags, and domains, as described earlier.

3.3 Preprocessing the Tweet

We are now ready to describe the ten main steps of our system. In the first step, we preprocess each incoming tweet. Specifically, we detect the language and drop the tweet if it is not in English. Next, we compute the Web context of the tweet, as described earlier. Next, we clean the tweet (e.g., removing user ID, emails, URLs, HTML tags), then tokenize it, using common separators such as white space, comma, period, colon, and semi-colon (currently we perform only limited spell correction on the tweet using a dictionary). Finally, since entity mentions are typically nouns or noun phrases (e.g., Obama, Hawaii), we perform lightweight part-of-speech tagging to detect nouns and noun phrases.

3.4 Extracting and Linking Entity Mentions

In the next step, we extract entity mentions from the tweet and link them to the nodes in the KB, whenever possible. For example, consider a toy KB with only three non-root nodes, n_1 = "Obama" (a person), n_2 = "Politics" (a topic), and n_3 = "Politics of Love" (a movie). Given the tweet "Politics of Love is about Obama's election", we want to generate the mentions

(Politics, n_2), (Politics of Love, n_3), (Obama, n_1).

The first mention, (Politics, n_2), for example says that string "Politics" in the tweet refers to node n_2 in the KB. (This mention is clearly incorrect, because "Politics" here is a part of "Politics of Love", a movie name. Subsequent steps can remove such incorrect mentions, as we show later.)

A Dictionary-Based Approach: Toward the above goal, we take a dictionary-based approach. That is, we parse the tweet to find strings that match the names of the nodes in the KB, then link the strings to those nodes to form entity mentions (later we show how to remove mentions that are incorrectly linked). In this sense, the node names form a giant dictionary against which we match the tweet.

To match efficiently, we construct a prefix map, which is a hash with entries of the form $(prefix, nodeID, stop?)$. Here $prefix$ is a token-level prefix of a node name in the KB. If $prefix$ happens to be the full name of a node, then $nodeID$ refers to the ID of that node. In this case, *stop*? is set to true if there is no other node that has $prefix$ as a prefix, and false otherwise. If $prefix$ is not the full name of a node, then $nodeID$ is set -1 and *stop*? is set to false.

For example, consider again the above toy KB of nodes $n_1 - n_3$. The prefix map for this KB consists of four entries: (Obama, n_1, t), (Politics, n_2, f), (Politics of, -1, f), and (Politics of Love, n_3, t). We omit details on how to construct the prefix map except to note that it is generated offline then loaded into memory (only once, at the start of our system).

Given a tweet, we parse it word by word and use the prefix map to locate matching strings. Consider again the tweet "Politics of Love is about Obama's election". By looking up the first word of the tweet, "Politics", in the prefix map, we can see that "Politics" is the full name of node n_2, but is still a prefix of some other node. So we generate the mention (Politics, n_2), but continue the search. Eventually we find that "Politics of Love" is the full name of node n_3 (thus generating a new mention), and that it is not a prefix of any other node. So the search *restarts* at the next word, "is", and so on. This algorithm scans each tweet only once, and thus is time efficient. As described, it does not consider different word orderings in the tweet (e.g., recognizing that "Barack Obama" and "Obama, Barack" refer to the same node in the KB).

Pros and Cons of the Dictionary-Based Approach: The dictionary-based approach is time-efficient and well-suited for tweets, which often contain broken ungrammatical sentence fragments. Nevertheless, it fails in certain cases. For example, given the tweet "watched Obama election tonite with Stephanie", the dictionary approach may be able to extract "Obama" and "Obama election". But it may not be able to extract "Stephanie" as a person name, if Stephanie is a relatively unknown person who is not in the KB. Put another way, the dictionary approach is limited to recognizing only entities that are in the KB.

To address this limitation, in this step we also employ off-the-shelf state-of-the-art named entity recognizers to parse tweets. Currently we only employ these recognizers to find person names (e.g., Stephanie), because we found that they could help improve our recall in recognizing person names. To scale, we note that we have a limited amount of time to process each tweet. So if the off-the-shelf recognizers cannot finish in time, we stop them and use only the output of the dictionary-based approach.

Homonyms and Synonyms: We note that extracted mentions can be (and often are) homonyms (i.e., having the same name but linking to different entities). For example, if a tweet mentions "apple", then the system will produce one mention that links "apple" to the node Apple Corp in the KB, and another mention that links "apple" to the node Apple (fruit). In general, the system will produce as many mentions as the number of homonyms we have in the KB. We show later in Step 3.9 how to disambiguate the mentions, i.e., removing all homonym mentions except one.

Synonyms (i.e., the same entity appearing under different names) are also often a problem in text processing. Our current system assumes that the synonyms are already in the KB, as a set of nodes that all point to a canonical node. For example, in the KB we collect as many synonyms as possible for President Obama (e.g., Obama, BO, Barack, etc.), creating nodes for them, and linking all of them to the canonical Wikipedia page of Barack Obama. This way, if a tweet mentions one of the synonyms, we can resolve it to the canonical entity.

3.5 Filtering and Scoring Mentions

The previous step generates a set of entity mentions. In this step we apply hand-crafted rules to drop certain mentions judged incorrect, then score the remaining mentions. Examples of these rules include:

- Overlapping mentions: For example, a rule may drop a mention if it is subsumed by another mention (e.g., mention (Politics, n_2) is subsumed by (Politics of Love, n_3)).
- Blacklists of strings and IDs: drop a mention if it is a

blacklisted string (e.g., stop words, profanities, slangs, etc.; about 127K strings have been blacklisted), or if the mention refers to a blacklisted node in the KB (about 0.5M nodes have been blacklisted).

- Prefix/suffix: drop a mention if it contains certain characters or words as prefixes or suffixes, or if the node name begins with a stop word (e.g., "a", "an", "the").

- Size/number: drop a mention if it is a one-letter word (e.g., "a", "b") or a number.

- Straddling sentence boundaries: drop a mention if it straddles sentence boundaries, such as mention "Indian Cricket teams" from tweet "He's an Indian. Cricket teams are popular in India."

- Part of a noun: if a mention is one word and the word before or after this mention is a noun phrase, then this mention is likely to be part of that noun phrase. Hence, drop this mention.

We use 14 such rules, which drop mentions by only examining the textual content of the tweet. Once we have dropped certain mentions, we score the remaining ones. (This is just a "quick and dirty" score; we compute a more complex score later.) Briefly, we consider whether the mention is a noun or noun phrase, whether it is a popular homonym (i.e., its Wikipedia page has a high traffic count in the past few days), and whether it appears in the title of any article referred to in the tweet. The more of these conditions are true, the more likely that the mention is indeed a valid mention (i.e., a mention of a real-world entity) that is linked to the right node in the KB. We omit describing the exact scoring function due to space reasons.

3.6 Tagging and Classifying the Tweet

In the next step we tag and classify the tweet. Again, this is just a "quick and dirty" tagging and classifying, whose results are used to generate features. These features are in turn used to evaluate the mentions. Later, in Step 3.11, after the mentions have been thoroughly evaluated and incorrect mentions removed, we redo the tagging and classifying step to obtain a more accurate result.

We now describe the tagging procedure. Let (m_1, n_1, s_1), $\ldots$, (m_q, n_q, s_q) be the q mentions that we have generated for the tweet in the previous steps. Here, a mention (m_i, n_i, s_i) means the string m_i in the tweet refers to node n_i in the KB, and has a score s_i.

We assume that the above mentions have been sorted in decreasing order of score. Consider the first mention (m_1, n_1, s_1) (i.e., the one with the highest score). Recall that n_1 is a node in the KB. Starting at n_1, we go up all lineages of n_1, all the way to the root, and assign to each node in these lineages a score (currently set to be s_1, the same score as that of n_1). For example, suppose that n_1 is "Forced Suicide" and that it has two lineages: Force Suicide - Ancient Greek Philosophers - Philosophers - ROOT, and Force Suicide - 5th Century BC Philosophers - Philosophers - ROOT. Then all nodes along these two lineages get assigned the same score as that of n_1.

The intuition is that since the tweet mentions n_1, it follows that all nodes in all lineages of n_1 are also relevant to the tweet. Hence, they all get assigned a non-zero score that is proportional to how relevant n_1 is to the tweet.

Next, we process the second mention (m_2, n_2, s_2). Similarly, all nodes in all lineages of n_2 get assigned the score s_2, *except* however the nodes that also appear in the lineages of the first mention. Intuitively, such nodes should be more relevant to the tweet (since they appear in the lineages of two entity mentions, not just one). So their scores should be a combination of the two scores s_1 and s_2. Setting such scores to be $s_1 + s_2$, however, tend to overestimate the importance of such nodes. So instead we set their scores to $s_1 + s_2 - s_1 \cdot s_2$, using a probabilistic interpretation.

We proceed similarly with the remaining mentions. At the end, all nodes in the lineages of the nodes $n_1, \ldots, n_q$ have been assigned non-zero scores, and those appearing in multiple lineages have higher scores, reflecting the intuition that these nodes can be good tags that describe the tweets. We then normalize the node scores.

Let $\mathcal{C}$ be the list of all nodes with non-zero normalized scores. Next, we select from $\mathcal{C}$ all topic nodes. Recall that we have defined 23 topics, which are 23 children of the ROOT node in the KB; we also refer to these topic nodes as *vertical* nodes, because each such node represents a subtree in the KB's taxonomy that is a vertical, such as health, technology, politics, travel, and so on.

We select the v topic nodes with the highest score (where v is a system parameter), then remove from $\mathcal{C}$ all nodes that are not descendants of these v topic nodes. The intuition here is that a tweet is typically just about a small set of topics (v in this case). So any node outside those topics is likely to be noise and should be removed.

Finally, we sort and return the names of the nodes in $\mathcal{C}$, alongwith their scores, as the tags of the tweet. Given these tags, it is relatively straightforward to infer a set of topics for the tweet. We omit further details for space reasons.

Optimizations: To further improve the accuracy of tagging and classification, we apply a set of optimizations to the above process. First, we ignore all nodes that are blacklisted (recall that about 0.5M nodes in the KB are blacklisted).

Second, we observe that we still tend to overestimate the score of a node that appears in the lineages of multiple mentions. So we stop increasing the score of a node after it has received contributions from k mentions (currently set to 3).

Finally, we observe that certain nodes are more suited to be tags than others (e.g., "travel"), and that certain nodes should probably never be used as tags (e.g., "average"). Consequently, we create two lists of nodes: bad nodes and good nodes. During the above score computation process, if a node of a lineage is in the good-node or bad-node list, then we boost or decrease its score by a certain amount, respectively.

3.7 Extracting Mention Features

In this step we extract a broad range of features for mentions. Later we use these features to disambiguate and score mentions. Broadly speaking, we extract features from tweet, the KB, and other external sources. Examples of features extracted from the tweet itself include:

- Similarity score between the mention and the tweet: Let n be the node referred to in the mention, and let S be the set of nodes that appear in the lineages of n as well as in the tags of the tweet, as computed in

Section 3.6 (recall that each tag is technically the name of a node in the KB). We compute a weighted sum of the scores of the nodes in S and return the sum as the similarity score between the mention and the tweet.

- Similarity scores between the mention and the social context of the user, the hashtags in the tweet (if any), and the Web domains in the tweet (if any).
- Does the mention appear in the title (of an article that the tweet refers to)?
- If the node of the mention is a descendant of a certain concept in the KB (e.g., Books, Comics, Movies, Films, Songs), does the mention begin with an uppercase character? This feature captures the intuition that often the name of a book, comic, movie, etc. begins with an uppercase character.
- Other examples include: the number of times the mention appears in the tweet, its position, is it part of a hashtag?

Examples of features extracted from the KB include: How many homonyms does this mention appear in (a measure of how ambiguous this mention is)? How many descendants does the node of the mention have? How many children? The depth of the node in the taxonomy? Is the node an instance or a concept? Is the node a person name? A location? An acronym?

Examples of features extracted from other sources include: the number of words in the mention that are capitalized, how popular is this mention? (For example, how often does it appear on the Web? How many times has its Wikipedia page been accessed in the past day, past week, past month?) How often is the mention linked in text as opposed to being unlinked? (If the mention is often linked, then it is likely to be a valid mention of a real-world entity.) How generic is the mention (e.g., "calm" and "collected" are too generic to be useful)? How often is it being searched for (collected using search logs)? What is its click-through rate?

Social Signals: As described, it is clear that we extract many social signals as features. Examples include Wikipedia traffic, traffic on a variety of social media sites (e.g., Pinterest), search frequency, and click-through rate. Together with social contexts, such signals help boost the processing accuracy.

3.8 Filtering the Mentions

Once we have extracted mention features, we apply a set of hand-crafted rules that use these features to remove certain mentions. An example rule may remove a mention if its node has fewer than 5 children. Another rule may remove a mention if the Wikipedia page associated with its node has received fewer than 10 visits in the past month. Yet another filter may remove a mention if its node is a descendant of "Books" and yet the mention does not begin with an uppercase character, and so on.

3.9 Disambiguating the Mentions

In the next step, we disambiguate mentions such as Apple the company vs. Apple the fruit. To do so, for each ambiguous mention (i.e., a mention with multiple homonyms), we compute a disambiguation score between its node and the tweet. This score considers the followings:

- the popularity of the node (mostly measured in the traffic to the Wikipedia page associated with the node);
- the similarity scores between the mention and the tweet, the mention and the user, the mention and the hashtags and the Web domains, as discussed earlier;
- a similarity score computed between the Web context of the node and the tweet; and
- a similarity score computed between the social context of the node and the tweet.

Clearly, the higher the disambiguation score, the more likely that the tweet refers to the concept (or instance) associated with the node. Hence, we select the highest-scoring node among the ambiguous nodes (assuming that its score exceeds a threshold). For example, suppose a tweet mentions "apple", resulting in two mentions to Apple the company and Apple the fruit, respectively. Suppose that the disambiguation scores of these two mentions are 0.8 and 0.5. Then we select the first mention, and thus the node referring to Apple the company, as the interpretation of the string "apple" in the tweet.

3.10 Scoring the Mentions Again

In this step we score the mentions again, for the last time. For each mention, we compute a score on how likely the mention is. Currently, the score is computed using a logistic regression function over a subset of the features generated in Section 3.7. This function is trained over a sample of manually curated data.

3.11 Tagging and Classifying the Tweet Again

At this point, we assume that all mentions have been thoroughly evaluated, most incorrect mentions have been removed, and the scores of the mentions have been computed as carefully as possible. So we use these mentions to tag and classify the tweet again, in the same way as we do in Section 3.6. This produces a revised set of tags and topics for the tweet.

3.12 Applying Editorial Rules

In the final step of the system, we apply editorial rules to clean the mentions, tags, and classification labels. For example, a rule may be !playbook & (blackberry | bberry) = Blackberry Mobile Phones. This rule says that if a mention does not contain "playbook" but does contain "blackberry" or "bberry" (ignoring cases), then link this mention to concept "Blackberry Mobile Phone" in the KB. In general, each rule is a pair $(regex, action)$, which means that if the mention matches the regular expression $regex$, then apply $action$. Our current system has 231 such rules.

After applying editorial rules, we output the final mentions, tags, and topics for the tweet, together with scores.

3.13 Discussion

As described, our approach to semantically processing a tweet has several distinguishing characteristics:

- First, we interleave the four tasks: extracting, linking, classifying, and tagging. As Sections 3.3-3.12 demonstrate, the goal of such interleaving is to use the output of one task to help another task. For example, a preliminary step of extracting and linking produces a

preliminary set of mentions. We use these mentions to help tagging and classifying tweets. The result in turn helps extracting and linking. And the result of this in turn helps tagging and classifying.

- Second, we use a lot of context information. These include Web context for tweets, social contexts for users, hashtags, and domains, and Web and social contexts for KB nodes. Since tweets (and many other types of social media data, such as Facebook updates) tend to be quite short, it is critical to generate and use contexts to improve the processing accuracy.
- Third, we use a lot of social information. This takes the form of social contexts, as described earlier, as well as social signals, such as traffic on social Web sites (e.g., Wikipedia, Pinterest), search frequency, and click-through rate. The social information helps us significantly boost the accuracy of the system.
- Fourth, since we have to process tweets in real time, and scale to 3000-6000 tweets per second, we do not use complex, opaque, or time intensive techniques in our online pipeline. (We do use some of these techniques in offline processing; see Sections 3.1-3.2.) An added benefit of this strategy is that it gives us a lot of fine-grained control over the entire pipeline. We have found this control capability to be critical in adapting the pipeline quickly to a new application and to sudden changes in the tweet stream. Another benefit is that we can quickly train a new developer to understand, use, debug, and maintain the pipeline, an important requirement in certain high-turnover situations.
- Finally, we use hand-crafted rules extensively, in several places of the pipeline. Our main conclusions are that it is possible to supply such rules even with a relatively small development team, and that the rules are important to improve the accuracy and to exert fine-grained control over the pipeline.

4. EMPIRICAL EVALUATION

We now describe experiments that evaluate our system, compare it to current approaches, and measure the utility of various components in our system.

Data Sets: We first sampled 500 English tweets from the firehose, and discarded a few bad tweets (e.g., non-English, containing only an URL) to obtain a sample of 477 tweets. Next, we manually identified all entity mentions in the tweets and linked them to the correct nodes in our KB. In total we identified 364 entity mentions.

Next, we manually identified the topics and the tags for each tweet. Doing this turned out to be highly time consuming and rather tricky. Since our KB is quite large, with 13+ million nodes, finding all good tags for a tweet is typically not possible. Thus, for each tweet, we ran our system with the lowest possible thresholds, to generate as many tags as possible. Next, we manually examined these tags to select the best tags for the tweet. Finally, we added any tag that we think the tweet should have, and yet was not produced by our system.

Given the time-consuming nature of the above process, so far we have manually tagged and assigned topics to 99 tweets. We use these 99 tweets to evaluate classification and tagging, and use the 477 tweets mentioned above to evaluate extraction and linking. We are in the process of expanding the above two evaluation sets further.

Entity types	P	R	F1
Overall	77.74	62.36	69.21
Person	84.31	71.07	77.13
Organization	85.29	61.70	71.60
Location	97.96	72.73	83.48
MedicalCondition	100.00	100.00	100.00
RadioProgram	100.00	100.00	100.00
SportsEvent	100.00	59.09	74.29
Movie	100.00	50.00	66.67
Position	62.50	35.71	45.45
TVShow	50.00	33.33	40.00
Product	38.46	38.46	38.46
Technology	50.00	16.67	25.00
MusicAlbum	16.67	33.33	22.22
Song	28.57	16.67	21.05
MusicEvent	0.00	0.00	0.00

Table 1: The accuracy of our system for extraction and linking

4.1 Measuring the Accuracy of Our System

Extraction and Linking: Table 1 shows the accuracy of our system for extraction and linking. Here P, shorthand for precision, is the fraction of mentions predicted by the system that is correct, in that the mention is a valid mention of a real-world entity and its linking to an entity in the KB (if any) is correct. R, shorthand for recall, is the fraction of mentions in the golden set of mentions that the system successfully retrieved, and $F_1 = 2PR/(P + R)$.

The first line ("Overall") of this table shows that our system achieves reasonably high accuracy, at 77.74% precision and 62.36% recall. We perform quite well on the common categories of person, organization, and location (as the next three lines show). Later we show that we do better here than current approaches.

The subsequent lines show our accuracy per various other entity categories. Here the results range from being perfect (e.g., Medical Condition, which tend to be long unique strings), to reasonable (e.g., Sport Events, Movies), to relatively low (e.g., Music Album, Song). In particular, we found that it is very difficult to accurately extract movie, book, and song names that are "generic", such as "It's Friday" or "inception". The problem is that such phrases commonly appear in everyday sayings, e.g., "Thank God it's Friday".

Classification: Table 2 shows the accuracy of our system for classification (i.e., how well we assign topics to tweets). The overall performance (first line) is respectable at 50% precision and 59.74% recall, with clearly ample room for improvement. The subsequent lines show a breakdown of accuracy per topic. Here again the performance ranges from perfect (e.g., Environment, Travel) to reasonable (e.g., Health)

Topics	P	R	F1
Overall	50.00	59.74	54.44
Environment	100.00	100.00	100.00
Travel	100.00	100.00	100.00
PoliticsAndGovernment	80.00	100.00	88.89
Health	80.00	80.00	80.00
ScienceAndNature	60.00	100.00	75.00
Technology	60.00	100.00	75.00
Sports	100.00	56.00	71.79
ArtsAndEntertainment	53.85	70.00	60.87
BusinessAndFinance	25.00	66.67	36.36
Products	15.38	66.67	25.00
SocialSciences	14.29	100.00	25.00
People	100	7.69	14.29
FoodAndAgriculture	-	0.00	-
HomeAndLeisure	-	0.00	-

Table 2: The accuracy of our system for classification task

to relatively low (e.g., Products, People). (Note that for the last two topics, our system did not make any predictions.)

Tagging: For the task of assigning descriptive tags to tweets, our system achieves 35.92% precision, 84.09% recall and 50.34% F_1.

4.2 Comparison with Existing Approaches

We compare our system with the off-the-shelf popular Stanford Named Entity Recognizer and the popular industrial system OpenCalais. Specifically, we consider three versions of the Stanford system:

- StanNER-3: This is a 3-class (Person, Organization, Location) named entity recognizer. The system uses a CRF-based model which has been trained on a mixture of CoNLL, MUC and ACE named entity corpora.
- StanNER-3-cl: This is the caseless version of StanNER-3 system which means it ignores capitalization in text.
- StanNER-4: This is a 4-class (Person, Organization, Location, Misc) named entity recognizer for English text. This system uses a CRF-based model which has been trained on the CoNLL corpora.

OpenCalais is an industrial product of Thomson Reuters which provides open APIs to extract entities, facts, events and relations from text and assign topics and tags to text. Currently, the system supports 39 entity types and 17 topics.

Comparing Accuracy for Person, Organization, and Location: Tables 3.a-d show the accuracy of our system (listed as "doctagger", a name used internally at WalmartLabs) vs the Stanford variants vs OpenCalais. Since the current Stanford variants focus on extracting person names, organizations, and locations, the table compares the accuracy only for these categories.

The tables show that our system outperforms the other two in almost all aspects, especially with respect to extracting organizations. A main reason for low precision in the other systems is that they interpret many interjections (rofl, lmao, haha, etc) and abbreviations as organization names. A main reason for low recall is the difficulty in recognizing an organization name without using a large KB. For example, most NER tools without a large KB would incorrectly identify "Emilie Sloan" as a person, not an organization.

(a) Overall measures on {PER,ORG,LOC}

Systems	P	R	F1
Doctagger	87.67	68.33	76.80
StanNER-3	78.67	41.99	54.76
StanNER-3-cl	71.17	41.28	52.25
StanNER-4	48.55	47.69	48.11
OpenCalais	78.43	28.47	41.78

(b) Measures on PER only

Systems	P	R	F1
Doctagger	84.31	71.07	77.13
StanNER-3	82.41	61.98	70.75
StanNER-3-cl	70.30	56.68	63.96
StanNER-4	74.07	66.11	69.87
OpenCalais	86.79	38.02	52.87

(c) Measures on ORG only

Systems	P	R	F1
Doctagger	85.29	61.70	71.60
StanNER-3	47.62	10.64	17.39
StanNER-3-cl	60.00	12.77	21.05
StanNER-4	21.01	26.60	23.47
OpenCalais	40.91	9.57	15.51

(d) Measures on LOC only

Systems	P	R	F1
Doctagger	97.96	72.73	83.48
StanNER-3	86.84	50.00	63.46
StanNER-3-cl	78.57	50.00	61.11
StanNER-4	59.18	43.94	50.43
OpenCalais	92.59	37.88	53.76

Table 3: Our system vs Stanford system variants vs OpenCalais in extracting persons, organizations, and locations

Systems	Entity extraction and linking			Classification			Tagging		
	P	R	F1	P	R	F1	P	R	F1
Doctagger	77.74	62.36	69.21	50.00	59.74	54.44	35.92	84.09	50.34
OpenCalais	70.67	29.12	41.25	50.00	32.47	39.47	40.80	-	-

Table 4: Our system vs OpenCalais for all tasks

Our System vs OpenCalais: OpenCalais is quite similar to our system, in that it can perform all four tasks of extraction, linking, classification, and tagging, and that it can handle a large number of categories (in contrast, the current Stanford variants only focus on extracting persons, organizations, and locations).

Thus, in the next step, we compare our system to OpenCalais. Table 4 shows the overall performance of the two systems for the tasks. (Note that for tagging, we do not have access to the internal KB used by OpenCalais, and hence are unable to compute recall, and thus F_1.) The table shows that we significantly outperform OpenCalais, except in the precision of tagging (35.92% vs 40.8%).

Table 5 shows a breakdown of the accuracy of entity extraction and linking per categories. In general, OpenCalais extracts relatively few entities, which explains its low recall. The table shows that our system outperforms OpenCalais in all categories except Position and Technology. (Note that for certain categories, our system or OpenCalais made no prediction; in such cases we compute recall to be 0.00.)

4.3 Evaluating Components' Utilities

Table 6 shows the utility of various components in our system. The rows of the table show the accuracy of the complete system, the system without using any context information, the system without using any social signal, and a baseline system, respectively. The baseline system simply extracts mentions using the names of the nodes in the KB, then matches each mention to the most popular homonym

Entity types	Doctagger			OpenCalais		
	P	R	F1	P	R	F1
Overall	77.74	62.36	69.21	70.67	29.12	41.25
Person	84.31	71.07	77.13	86.79	38.02	52.87
Organization	85.29	61.70	71.60	40.91	9.57	15.52
Location	97.96	72.73	83.48	92.59	37.88	53.76
MedicalCondition	100.00	100.00	100.00	33.33	25.00	28.57
Movie	100.00	50.00	66.67	0.00	0.00	0.00
SportsEvent	100.00	59.09	74.29	100.00	45.45	62.50
Product	38.46	38.46	38.46	100.00	15.38	26.67
RadioProgram	100.00	100.00	100.00	-	0.00	-
TVShow	50.00	33.33	40.00	-	0.00	-
MusicAlbum	16.67	33.33	22.22	-	0.00	-
Song	28.57	16.67	21.05	-	0.00	-
MusicEvent	-	0.00	-	-	0.00	-
Position	62.50	35.71	45.45	66.67	71.43	68.97
Technology	50.00	16.67	25.00	60.00	50.00	54.55

Table 5: Our system vs OpenCalais for entity extraction and linking tasks

Systems	Entity extraction and linking			Classification			Tagging		
	P	R	F1	P	R	F1	P	R	F1
Doctagger	77.74	62.36	69.21	50.00	59.74	54.44	35.92	84.09	50.34
Doctagger w/o context	74.18	62.36	67.76	47.83	28.57	35.77	31.44	69.32	43.26
Doctagger w/o social signals	51.68	54.95	53.26	34.48	51.95	41.45	20.18	76.14	31.90
Baseline	19.23	28.85	23.08	13.68	20.78	16.99	4.49	27.27	7.71

Table 6: The accuracy of our system as we turn off various components

node (i.e., the one with the most traffic).

The table shows that the baseline system achieves very low accuracy, thereby demonstrating the utility of the overall system. The table also shows that accuracy drops significantly without using contexts or social signals, thereby demonstrating the utility of these components in our system.

4.4 Summary

The experiments show that our system significantly outperforms current approaches. However, there is still ample room for improvement, especially in certain categories such as Product and Music. The experiments also show that using contexts and social signals is critical to improving accuracies.

5. APPLICATIONS & LESSONS LEARNED

Our system has been used extensively at Kosmix and later at WalmartLabs in a variety of applications. We now briefly describe some of these applications, and the lessons learned.

5.1 Sample Applications

Event Monitoring in the Twittersphere: In late 2010 Kosmix built a flagship application called Tweetbeat that monitors the Twittersphere to detect interesting emerging events (e.g., Egyptian uprising, stock crash, Japanese earthquake), then displays all tweets of these events in real time.

Figure 3: Event monitoring in social media using Tweetbeat

Figure 3 shows an example. This was a little widget embedded on the ABC news homepage, and powered by Kosmix. For the event "Egyptian uprising" (which was probably the hottest political event at that time), the widget shows interesting tweets related to that event, scrolling in real time.

To do this, for each incoming tweet we must decide whether it belongs to an event E. The technique used to decide this is complex and will be described in an upcoming paper. However, an important component of this technique is that we perform extraction, linking, classification, and tagging of each incoming tweet. If the tweet refers to nodes in the KB that we already know are involved in the event E, then we know that the tweet is more likely to refer to event E.

In-context Advertising: The basic idea here is that when a user is reading a Web page (e.g., a newsarticle or a discussion forum page), we parse the page, identify the most important concepts/instances on the page, and then highlight those concepts/instances. When the user hovers over a highlight, he or she will see a pop-up ad that is relevant to the highlighted concepts/instances. To identify the most important concepts/instances on the page, we use the system described in this paper.

Understanding User Queries: Kosmix started out as a Deep Web search engine. A user poses a search query such as "Las Vegas vacation" on kosmix.com, we interpret the query, go to an appropriate set of Deep Web data sources (i.e., those behind form interfaces), query the sources, obtain and combine the answers, then return these to the user.

Clearly, understanding the user query is a critical step in the above process. To understand a query, we use the current system to detect if it mentions any concept or instance in the KB (sort of treating the user query as a short tweet).

Product Search: In mid 2011 Kosmix was acquired by Walmart and since then we have used the above system

to assist a range of e-commerce applications. For example, when a user queries "Sony TV" on walmart.com, we may want to know all categories that are related to this query, such as "DVD", "Bluray players", etc. We use the current system to find such related categories.

Product Recommendation: In spring 2012 we introduced a Facebook application called ShopyCat. After a Facebook user installs this application and gives it access to his/her Facebook account, the application will crawl his/her posts as well as those of his/her friends, then infer the interests of each person. Next, the application uses these interests to recommend gifts that the user can buy for his or her friends. For example, ShopyCat may infer that a particular friend is very interested in football and Superbowl, and hence may recommend a Superbowl monopoly game from deluxegame.com as a gift.

ShopyCat infers the interests of a person by processing his or her posts in social media using the current system, to see what concepts/instances in our KB are frequently mentioned in these posts. For example, if a person often mentions coffee related products in his or her posts, then ShopyCat infers that he or she is likely to be interested in coffee.

Social Mining: In one of the latest applications, we use our system to process all tweets that come from a specific location, to infer the overall interests of people in that location, then use this information to decide how to stock the local Walmart store. For example, from mining all tweets coming from Mountain View, California, we may infer that many people in Mountain View are interested in outdoor activities, and so the outdoor section at the local Walmart is expanded accordingly. Such social mining appears to be quite useful on a seasonal basis.

5.2 Lessons Learned

We have found that it is possible to use a modest team (of no more than 3 full-time persons at any time; this does not count the team that builds the KB) to build an end-to-end system that semantically processes social data and pushes the current state of the art. In this system, social signals and contexts play an important role in achieving good accuracy. We also found that even when the accuracy is not perfect, the system already proves useful for a variety of real-world applications.

There is still ample room for improvement in terms of accuracy, however. For example, as discussed in Section 4.1, we found that it is very difficult to accurately extract movie, book, and song names that are "generic", such as "It's Friday" or "inception", because they commonly appear in everyday tweets, e.g., "Thank God it's Friday". How to detect such tweets and avoid extracting a false positive is a challenging problem.

6. RELATED WORK

Entity extraction and classification of formal text has been widely studied for more than two decades, in both the database and AI communities. A variety of techniques ranging from hand-coded rules to statistical machine learning has been proposed. Fastus [4], Circus [23], DBLife [12] and Avatar [19] are examples of systems based on hand-coded rules. Systems such as Aitken [2], Califf and Mooney [9], and Soderland [34] learn rules automatically. Statistical learning based systems have used a variety of approaches, such as hidden Markov models [1, 8, 33], maximum entropy [21, 26, 29] and conditional random fields [22]. A recent survey [31] discusses state-of-the-art information extraction techniques in depth.

Competitions such as CoNLL [38], MUC [37] and ACE [15] made available large annotated corpora of news articles, thereby fostering the growth of many commercial tools. Examples include Stanford NER [17], OpenNLP [6], GATE [11], and LingPipe [24]. Stanford NER, based on CRF classifiers, is a popular and state-of-the-art tool for extraction of entities from formal text.

Entity extraction and classification for tweets, on the other hand, has been a less studied problem. Liu et al. [25] present a semi-supervised solution that combines a KNN and a CRF classifier. They also use several gazetteer lists covering common names, companies, locations, etc. Their use of gazetteer lists resembles our use of a KB. However, their solution extracts only person, organization and location entities, while we do it for a large number of entity types with links to our KB. Finn et al. [16] use crowdsourcing solutions to annotate a large corpus of tweets to create training data. Recently Ritter et al. [30] have developed a NLP pipeline spanning POS tagging, chunking, named entity recognition and classification for tweets. Han and Baldwin [18] have worked on normalization of tweets to improve the sentence structure which could potentially help any semantic task on tweets. For example, our system too can use normalization as a preprocessing step.

DBpedia [5], YAGO [35], Freebase [7] and Google knowledge graph are well-known examples of KBs constructed using information extraction (IE) techniques. While IE techniques can help extend existing KBs [36], KBs in turn can help improve IE [10]. Our system uses a KB (we describe building our KB in [13]) to extract and link entities and concepts. Our work is similar to Wikify! [27] which extracts important concepts from text and links them to Wikipedia pages. However, our KB is richer than Wikipedia and we use a lot of social and Web contexts and signals to handle social media text. Wiki3C [20] is another related work that ranks the Wikipedia categories of a concept extracted from text based on the context.

As far as large-scale systems are concerned, SemTag and Seeker [14] is one of the earlier attempts at performing automatic semantic tagging of a large Web corpus using an ontology. Currently there are many industrial systems such as OpenCalais [28], AlchemyAPI [3], Semantria [32] performing large scale entity extraction and classification. Semantria does not support linked data whereas OpenCalais and AlchemyAPI do. OpenCalais additionally extracts relations, facts and events. AlchemyAPI provides APIs for language identification, sentiment analysis, relation extraction, and content scraping. We evaluated OpenCalais as it seems to be the market leader providing services to sites such as CNET and The Huffington Post.

7. CONCLUDING REMARKS

In this paper we have described an end-to-end industrial system that performs entity extraction, linking, classification, and tagging for social data. To the best of our knowledge, this is the first paper that describes such a system in depth. We have presented experiments that show that our system significantly outperforms current approaches, though it still has plenty of room for improvement. We have showed

that while not perfect, the system has proved useful in a variety of real-world applications. Finally, we have also demonstrated that it is important to exploit contexts and social signals to maximize the accuracy of such systems. We are currently working to open source a version of this system, for research and development purposes in the community.

8. REFERENCES

[1] E. Agichtein and V. Ganti. Mining reference tables for automatic text segmentation. In *SIGKDD*, 2004.

[2] J. S. Aitken. Learning information extraction rules: An inductive logic programming approach. In *ECAI*, 2002.

[3] AlchemyAPI. http://www.alchemyapi.com/.

[4] D. E. Appelt, J. R. Hobbs, J. Bear, D. Israel, and M. Tyson. FASTUS: A finite-state processor for information extraction from real-world text. In *IJCAI*, 1993.

[5] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A nucleus for a web of open data. In *The Semantic Web*, 2007.

[6] J. Baldridge. The OpenNLP project, 2005.

[7] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *SIGMOD*, 2008.

[8] V. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records. In *SIGMOD Record*, 2001.

[9] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *AAAI*, 1999.

[10] W. W. Cohen and S. Sarawagi. Exploiting dictionaries in named entity extraction: combining semi-Markov extraction processes and data integration methods. In *SIGKDD*, 2004.

[11] H. Cunningham, K. Bontcheva, and D. Maynard. GATE: an architecture for development of robust HLT applications. In *ACL*, 2002.

[12] P. DeRose, W. Shen, F. Chen, A. Doan, and R. Ramakrishnan. Building structured web community portals: A top-down, compositional, and incremental approach. In *VLDB*, 2007.

[13] O. Deshpande, D. S. Lamba, M. Tourn, S. Das, S. Subramaniam, A. Rajaraman, V. Harinarayan, and A. Doan. Building, maintaining, and using knowledge bases: A report from the trenches. In *SIGMOD*, 2013.

[14] S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien. SemTag and Seeker: Bootstrapping the semantic web via automated semantic annotation. In *WWW*, 2003.

[15] G. Doddington, A. Mitchell, M. Przybocki, L. Ramshaw, S. Strassel, and R. Weischedel. The automatic content extraction (ACE) program–tasks, data, and evaluation. In *LREC*, 2004.

[16] T. Finin, W. Murnane, A. Karandikar, N. Keller, J. Martineau, and M. Dredze. Annotating named entities in Twitter data with crowdsourcing. In *HLT-NAACL*, 2010.

[17] J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. In *ACL*, 2005.

[18] B. Han and T. Baldwin. Lexical normalisation of short text messages: Makn sens a# twitter. In *ACL-HLT*, 2011.

[19] T. Jayram, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu. Avatar information extraction system. *IEEE Data Eng. Bull.*, 29(1):40–48, 2006.

[20] P. Jiang, H. Hou, L. Chen, S. Chen, C. Yao, C. Li, and M. Wang. Wiki3C: Exploiting Wikipedia for context-aware concept categorization. In *WSDM*, 2013.

[21] D. Klein and C. D. Manning. Conditional structure versus conditional estimation in NLP models. In *EMNLP*, 2002.

[22] J. Lafferty, A. McCallum, and F. C. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[23] W. Lehnert, J. McCarthy, S. Soderland, E. Riloff, C. Cardie, J. Peterson, F. Feng, C. Dolan, and S. Goldman. UMass/Hughes: Description of the CIRCUS system used for MUC-5. In *MUC-5*, 1993.

[24] LingPipe. http://alias-i.com/lingpipe/.

[25] X. Liu, S. Zhang, F. Wei, and M. Zhou. Recognizing named entities in tweets. In *ACL-HLT*, 2011.

[26] A. McCallum, D. Freitag, and F. Pereira. Maximum entropy Markov models for information extraction and segmentation. In *ICML*, 2000.

[27] R. Mihalcea and A. Csomai. Wikify!: Linking documents to encyclopedic knowledge. In *CIKM*, 2007.

[28] OpenCalais. http://www.opencalais.com/.

[29] A. Ratnaparkhi. Learning to parse natural language with maximum entropy models. *Machine learning*, 34(1-3):151–175, 1999.

[30] A. Ritter, S. Clark, and O. Etzioni. Named entity recognition in tweets: An experimental study. In *EMNLP*, 2011.

[31] S. Sarawagi. Information extraction. *Foundations and Trends in Databases*, 1(3):261–377, 2008.

[32] Semantria. https://semantria.com/.

[33] K. Seymore, A. McCallum, and R. Rosenfeld. Learning hidden Markov model structure for information extraction. In *AAAI*, 1999.

[34] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine learning*, 34(1-3):233–272, 1999.

[35] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, 2007.

[36] F. M. Suchanek, M. Sozio, and G. Weikum. SOFIE: A self-organizing framework for information extraction. In *WWW*, 2009.

[37] B. M. Sundheim and N. A. Chinchor. Survey of the message understanding conferences. In *ACL-HLT*, 1993.

[38] E. F. Tjong Kim Sang and F. De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *CoNLL*, 2003.