TQEL: Framework for Query-Driven Linking of Top-K Entities in Social Media Blogs

Abdulrahman Alsaudi
University of California, Irvine
alsaudia@uci.edu

Yasser Altowim
Saudi Data And Artificial Intelligence Authority
ytowim@nic.gov.sa

Sharad Mehrotra
University of California, Irvine
sharad@ics.uci.edu

Yaming Yu
University of California, Irvine
yamingy@uci.edu

ABSTRACT

Social media analysis over blogs (such as tweets) often requires determining top-k mentions of a certain category (e.g., movies) in a collection (e.g., tweets collected over a given day). Such queries require entity linking (EL) function to be executed that is often expensive. We propose TQEL, a framework that minimizes the joint cost of EL calls and top-k query processing. The paper presents two variants - TQEL-exact and TQEL-approximate that retrieve the exact / approximate top-k results. TQEL-approximate, using a weaker stopping condition, achieves significantly improved performance (with the fraction of the cost of TQEL-exact) while providing strong probabilistic guarantees (over 2 orders of magnitude lower EL calls with 95% confidence threshold compared to TQEL-exact). TQEL-exact itself is orders of magnitude better compared to a naive approach that calls EL functions on the entire dataset.

PVLDB Reference Format:
Abdulrahman Alsaudi, Yasser Altowim, Sharad Mehrotra, and Yaming Yu. TQEL: Framework for Query-Driven Linking of Top-K Entities in Social Media Blogs. PVLDB, 14(11): 2642 - 2654, 2021.
doi:10.14778/3476249.3476309

1 INTRODUCTION

Social media blogs usually contain ambiguous *mentions* that could potentially refer to real-world *entities*. In this paper, we study how top-k queries in the context of social media blogs can efficiently be evaluated. Given a collection of social media blogs $\mathcal{T}$ that contain a number of mentions, the goal is to identify the top-k real-world entities that are mentioned the most in $\mathcal{T}$. Consider, for example, a user creates a collection of tweets (by sampling the public Twitter API and/or by running keyword/phrase queries using Twitter's query interface [24, 29]) and would like to characterize $\mathcal{T}$ based on the top-k entities of a certain category – e.g., top-k "movies", top-k "athletes", or top-k "locations". If the text/metadata in the tweets explicitly identified real-world entities, we could lookup the associated categories in knowledge bases such as DBpedia [5] or Wikipedia [2] to appropriately tag the tweets with the corresponding categories. Then, finding the top-k entities in $\mathcal{T}$ in the context of the category of interest (e.g., movies, politicians, athletes) would be straightforward; we would simply count the number of times an entity corresponding to the category of interest is mentioned in $\mathcal{T}$, and choose the k entities with the highest counts.

However, entities are not explicitly associated with the tweets. Instead, entity extraction and lookup functions [30] are used to determine them. Such functions take as input the set of words, as well as, metadata associated with the tweet, and return a set of a sequence of words (referred to as a mention) that could correspond to real-world entities [14, 30]. For instance, a lookup function applied to a tweet *"Black Panther won an Oscar!"* may identify two mentions "Black Panther" and "Oscar". Such mentions rarely correspond to a unique real-world entity. The study in [22] shows that each mention, on an average, could correspond to 13.1 real world entities, each of which is associated with different categories. For instance, the "black panther" mention could either refer to a movie or an animal corresponding to the "movies" or "animals" categories respectively. Before the top-k entities within a given category can be identified, we must first disambiguate such mentions.

Techniques to disambiguate entity mentions have been extensively studied over the past decade. Such functions use a variety of approaches including machine-learning techniques and supervised learning algorithms [16, 38], comparing mentions in tweets bodies against an external knowledge base [13, 14, 19, 30, 31], (e.g., Wikipedia), or using the feedback of people as in crowdsourcing [10]. Once mentions have been linked to the correct entities, the top-k entities in the tweet collection within the category of interest can be easily determined.

The challenge in implementing top-k queries arises since entity linking functions are expensive. As such, applying it to the entire collection requires significant computation, leading to long latency in the results. Moreover, such computation is also wasteful since it requires linking mentions that are simply not of interest, i.e., those that are clearly not part of the top-k results. One strategy to overcome the challenge is to simply associate the mention with every possible entity rather than running the entity linking function. For example, "black panther" in the above tweet will be associated with the "movie" entity and the "animal" entity as well. Then we can simply return the top-k set after aggregating the number of occurrences for each entity. For instance, for the tweets shown in Table 1 (represented in Figure 1), a query for top-2 movies will return as a result: "Black Panther (2018 movie)" & "Black Panther (1977 movie)" since each of them has 4 occurrences. Clearly, the above strategy results in erroneous answers since the right response in

Proceedings of the VLDB Endowment, Vol. 14, No. 11 ISSN 2150-8097.
doi:10.14778/3476249.3476309

this example should have been "Black Panther (2018 movie)" & "Beautiful Creatures" had we fully disambiguated all the mentions. When adopting the same strategy for top-10 movies, top-10 politicians and top-10 locations on our dataset, the average precision of the results were 0.16 while the rank distance was 377.9! We explain how to measure precision and rank distance in Section 6.

In this paper, we propose TQEL (Top-k Query processing using Entity Linking), a framework that exploits the query semantics for adaptive application of entity linking to only a subset of the mentions that are required to answer the query. The TQEL framework can be invoked to answer the top-k query exactly. The resulting implementation, referred to as TQEL-exact, improves upon the naive strategy of fully linking all mentions (prior to query execution), to linking only a small subset that could determine the top-k results. TQEL-exact returns exactly the same answer as would be returned by the naive strategy, though at a fraction of the expense. While TQEL-exact improves upon the naive mechanism, it, nonetheless, incurs overheads specially when there are a large number of entities with possible frequency counts that are large and also close to each other in value competing to be in the top-k spots. As such, TQEL-exact is not suitable for queries that require faster responses. The main contribution of the paper is an approximate approach which we refer to as TQEL-approximate.

TQEL-approximate, instead of continuing to link entities until it guarantees that it has found a top-k result, stops much earlier as soon as it can establish that the entities in top-k result, it has found so far, have a probability of being in the answer above a user specified threshold τ. To achieve this, TQEL-approximate uses two statistical models to efficiently deliver the answer. First, it estimates the number of occurrences of each entity by applying normal approximation statistics on the distribution of mentions associated with a given entity where such an estimation can be computed efficiently. Based on the normal approximation step, TQEL-approximate then decides whether to link mentions further, or whether it expects that a *potential* answer to the top-k query has been found. It then invokes the validation step which uses Monte-Carlo simulation technique that generates N samples of the possible worlds and then calculates the probability of each entity being in the top-k using the N samples. If verification fails, TQEL-approximate performs more entity linkings until the verification step's stopping condition is met. Note that normal approximation estimation provides a fast mechanism to predict that an answer has been found, however, it is not used to verify answers but rather the Monte-Carlo simulation is used for that purpose. Additionally, TQEL doesn't require the distribution of mentions' linking probabilities to be normally distributed although the filter works best when they are. In effect, TQEL-approximate allows users to trade quality with latency. Our results over different data sets show that TQEL-approximate achieves an order of magnitude improvement over TQEL-exact – it finds top-k answers with confidence as high as 95% within 5-10 seconds, while an exact approach takes 100-300 seconds for the same query on the same machine.

In summary the main **contributions** of this paper are: **(a)** a framework that uses entity linking to evaluate top-k queries efficiently (Sections 2 & 3), **(b)** two heuristic approaches that return an exact answer for the top-k queries (Section 4) (TQEL-exact), **(c)** an approximate approach that relaxes the quality of the result by returning a top-k answer with probabilistic guarantees (Section 5). (TQEL-approximate), **(d)** Experimental evaluation of TQEL extensively using three datasets. (Section 6).

2 PRELIMINARIES

In this section, we will present the needed preliminaries that form the basis for the TQEL framework.

2.1 Dataset and Required Functions

Social Media Datasets. Let $\mathcal{T}$ be a collection of tweets $t_1, t_2, ..., t_n$, M be the set of mentions in tweet t_i's text $m_1^i, m_2^i, ..., m_{|M|}^i$, E be the set of entities $e_1, e_2, ..., e_{|E|}$. and each entity e_x is associated with one or more category $c_1, c_2, ..., c_{|C|}$. Tweets, in general, contain the text along with metadata about the tweeter and the tweet itself such as username, timestamp and location of tweet. We identify a word or sequence of words in the body of the text as a mention m_j^i if it could *potentially* be linked to real world entity e_x. These entities could be of any category such as movies, people or locations. At the ingestion time, tweets will undergo a prepossessing step in order to extract and normalize the text in the tweet. This is executed by removing the unnecessary words in the body of the tweet such as stop words and Twitter's special notation (e.g., "rt").

Query Model. We model our top-k query Q^k where it is executed on top of the tweets collection $\mathcal{T}$. Each query Q^k contains a category filter c_g such as people or locations. The main concept is to find k entities, that are associated with category c_g, mentioned the most in the tweets collection $\mathcal{T}$. As an example, an editor who is working for a magazine that rates and reviews movies is asked to provide a list of the top-2 movies mentioned in a given tweet collection $\mathcal{T}$. The query Q^2 is to find 2 entities mentioned in $\mathcal{T}$ where entities are associated with the category movies and the number of times they were mentioned in $\mathcal{T}$ is larger than the rest of entities. This query can be evaluated using our query model.

Entity Extraction and Lookup. An entity extraction and lookup function LU(t_i) takes tweet t_i and returns a set of 3-tuples {<m_j^i, e_x, $p(m_j^i, e_x)$>} where m_j^i is the j_{th} mention in the tweet t_i, e_x is a possible entity for the mention and $p(m_j^i, e_x)$ is the probability m_j^i links to e_j. The sum of all probabilities for a specific mention m_1^i is 1. For example, let t_1 be "Black Panther won an oscar!", LU(t_1) returns <m_1^1, Black Panther (2018)>, 0.74), <m_1^1, Black Panther (Animal)>, 0.12>, <m_1^1, Black Panther (1977)>, 0.10>, <m_1^1, OTHER>, 0.02>. where OTHER corresponds to linking m_1^1 to no entity.

This process is known as the candidate list generation for a specific mention. Most of the approaches rely on name dictionary based techniques [12, 14, 16, 31] which generates a map where keys are a list of all possible mentions and values are a set of entities that could potentially refer to the key. This limits the generation of mentions to only the the ones that are stored in the map. Another approach uses search engines [17, 27] as the vehicle for finding candidate entities where they return the top hits for each mention.

Entity Linking. Given a tweet t_i, mention m_j^i and a linking function (EL), the entity linking function links mention m_j^i in t_i with a real-world entity e_x. For example, Given the tweet "Black Panther won an oscar!", and the mention "Black Panther", EL will link the mention "Black Panther" to Black Panther (2018) entity.

Table 1: Sample of prepossessed tweets. Crossed sequence of words are dropped in the reprocessing step. Bolded sequence of words represent a mention that links to an entity. Underlined mentions refers to an entity from movies category.

TweetID	Tweeter	Text	Time	Location
t_1	u_1	**Black panther** *has* finally grossed $700 million domestically!	ts_1	l_1
t_2	u_2	**Emmy Rossum** ~~in~~ **Beautiful Creatures** ~~is~~ stunning	ts_2	l_2
t_3	u_3	**Pixar** makes the best animated movies	ts_3	l_3
t_4	u_4	**La La Land** best movie ~~of all~~ time!	ts_4	l_4
t_5	u_5	Another live reading ~~of a~~ random book ~~in my tbr~~ pile, **Beautiful Creatures**	ts_5	l_5
t_6	u_1	**Black Panther** won an **oscar!**	ts_6	l_1
t_7	u_6	~~A lot of~~ athlete will ~~be in~~ **space jam 2**	ts_7	l_6
t_8	u_7	Gonna watch **stardust** ~~and~~ **beautiful creatures**	ts_8	l_7
t_9	u_8	~~a~~ **black panther** ~~was~~ photographed ~~in~~ **Africa** last week	ts_9	l_8
t_{10}	u_3	**Emma Stone** ~~was~~ amazing ~~in~~ **la la land**	ts_{10}	l_9
t_{11}	u_9	**Marvel** made **michael jordan** famous ~~in~~ **black panther**	ts_1	l_{10}

The entity linking function is considered a cost-heavy function that require complex computations to achieve the sought precision when linking a mention in the text of a tweet with an entity. Some entity linking functions might also require analyzing and linking entities of the previous tweets from the same user along with any tweets from the same location and time in order to accurately pinpoint the mentioned entity [14]. Most of the techniques takes advantage of linking the mentions to target entities in a known knowledge base (e.g. Wikipedia) [13, 14, 31, 32] by leveraging the Wikipedia topology and the Wikipedia articles along with the text of the tweet. Other techniques use a crowd sourcing approach in order to link the identified mentions [10] to an entity. Other approaches adopt a machine learning approach where supervised and semi-supervised learning algorithms are run on top of annotated data corpses to find best candidate entity for each mention [38].

Since the entity linking functions require complex computations to get the best candidate, it is considered a bottleneck in any query processing system that relies on entity linking to answer the asked query. In our setup, we consider the functions LU(t_i) and EL(t_i, m_j^i) black box functions that could be replaced with any entity extraction, lookup and linking functions.

2.2 Exact Top-k Definitions

Answer Semantics. Let E_{c_g} be all the possible entities that are associated with category c_g and let $count(e_x)$ be the number of occurrences of e_x in $\mathcal{T}$ as identified by the entity linking function.

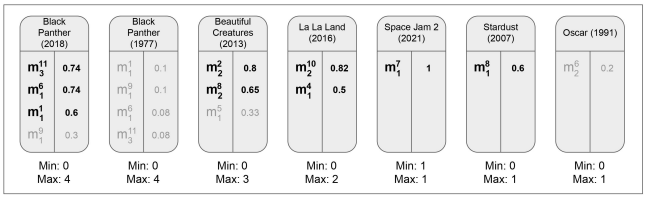

Figure 1: Entity list representation of Table 1. Bolded mentions represent mentions that links to associated entity. Faded mentions do not link to the associated entity.

Let $A(Q^k)$ be the answer of applying Q^k over $\mathcal{T}$. We call $A(Q^k)$ a valid answer iff:

- $A(Q^k) \subset E_{c_g}$.
- $\forall\, e_x \in A(Q^k)$, e_x appears only once in $A(Q^k)$.
- $\forall\, e_x \in A(Q^k) : \nexists\, e_y \in (E_{c_g} - A(Q^k))$, s.t. $count(e_y) > count(e_x)$

Given a query Q^2 looking for the top-2 movies in tweets in Table 1 and represented in Figure 1, answer $A(Q^k)$ = {Black Panther (2018), Beautiful Creatures (2013)} and answer $A(Q^k)$ = {Black Panther (2018), La La Land (2016)} are both valid.

Problem Definition. Given a top-k query Q^k on top of a collection of tweets $\mathcal{T}$, an entity lookup function LU, an entity linking function EL and a category c_g, efficiently generate an answer $A(Q^k)$ that satisfies the answer semantics of the exact approach .

Standard Solution. To generate a valid answer for Q^k over $\mathcal{T}$ for a specified category c_g, the standard solution first links each mention in $\mathcal{T}$. Entities that are associated with category c_g are filtered then the k entities with largest number of mentions are returned.

This approach requires the query executor to call the linking function on every mention in $\mathcal{T}$ which is inefficient. In Section 4, we will describe more efficient algorithms to compute the top-k answers that exploit query semantics to significantly reduce the number of entity linking functions invoked.

2.3 Approximate Top-k Definitions

Possible Worlds. A possible world w_a in our context consists of an assignment of each ambiguous mention to one of the possible entities based on its linking probability. Thus, the probability of the possible world p(w_a) can be computed as a product of such assignments. For example, world w_1 can consist of the following assignments based on Figure 1, {(m_3^{11}: Black Panther (2018)), (m_1^6: Black Panther (2018)), (m_1^1: Black Panther (1977)), (m_1^9: OTHER), (m_2^2: Beautiful Creatures (2013)), (m_2^8: Beautiful Creatures (2013)), (m_1^5: OTHER), (m_2^{10}: La La Land (2016)), (m_1^4: OTHER), (m_1^7: Space Jam 2 (2021)), (m_1^8: Stardust (2007)), (m_2^6: OTHER)}. p(w_1) $\approx$ 0.0022.

Probabilistic Top-k Evaluation. Given a possible world $w_a \in \mathcal{W}$, we evaluate the top-k query on the world w_a. Let $ans(w_a)$ be the top-k answer set for w_a, then $p(e_x \in$ top-k answers of ($\mathcal{W}$)),

abbreviated $tkp(e_x)$, can be calculated using the following equation:

$$tkp(e_x) = \sum_{a=1}^{|\mathcal{W}|} p(w_a) \cdot I(e_x, w_a) \quad (1)$$

where $I(e_x, w_a) = 1$ if $e_x \in ans(w_a)$, otherwise $I(e_x, w_a) = 0$

Answer Semantics. The approximate and the exact approaches share the same semantics with the change of third criteria to:

- $\forall\ e_x \in A(Q^k),\ \text{tkp}(e_x) > \tau$.

where τ is a user-defined confidence.

Our top-k query evaluation semantic is similar to PT-k [18] where only entities that have a top-k probability higher than a specific threshold will be included in the answer. The order among the returned top-k answer is ignored in our settings. Given the same query Q^2 that looks for the top-2 movies in tweets in Table 1, A possible answer for $A(Q^2)$ could be {Black Panther (2018), Beautiful Creatures (2013)} after the EL function has been called on some mentions. Note that there could be more than k entities that satisfy the answer semantics, however choosing any k entities from the full answer set is sufficient and correct.

Problem Definition. Given a top-k query Q^k on top of tweets collection $\mathcal{T}$, an entity lookup function LU, an entity linking function EL, a category c_g and a threshold value τ, we efficiently provide an approximate solution $A(Q^k)$ for Q^k that satisfies the approximate approach answer semantics.

2.4 Top-k Example Solution

Following the exact top-k definition, Q^2 for finding the exact top-2 movies in Table 1, the EL function is called on the following mentions set $\{m_3^{11}, m_1^6, m_2^2, m_2^8\}$. We can safely return the set {Black Panther (2018), Beautiful Creatures} as the answer without linking any more mentions as the answer satisfies the answer semantics. To solve the same query using the approximate approach where τ = 0.9, the calculation of $tkp(e_x)$ for all e_x is required. In our example the top-k probabilities are {(Black Panther (2018): 0.93), Beautiful Creatures (2013): 0.84), La La Land (2016), Space Jam 2 (2021): 0.27), (Stardust (2007): 0.16), (Black Panther (1977): 0.13), (Oscar (1991): 0.05)}. After calling the EL function on $\{m_2^2\}$, the top-k probabilities become {(Black Panther (2018): 0.93), Beautiful Creatures (2013): 0.91), ..}. Hence, we stop the execution and report the answer for Q^2 as {Black Panther (2018), Beautiful Creatures} since it satisfies the approximate solution's answer semantics.

3 TQEL OVERVIEW

Algorithm 1 describes the abstract description of the TQEL approach for linking entities while evaluating top-k queries. We will in the following sections refine the above abstraction to specify TQEL-exact (that identifies the true top-k results), and TQEL-approximate that identifies top-k results with probabilistic guarantees. TQEL framework consists of multiple phases where the first phase acts as a preparatory step for the mentions to be linked to entities. It also consists of a thinking phase that generates the execution plan for the current iteration and an act phase that executes the plan.

Algorithm 1 TQEL Approach

```
1:  procedure GetTopK(Q^k, T, LU, EL)
2:      E_LIST ← {}
3:      for each t ∈ T do
4:          {(m_j^i, e_x, p)} ← LU(t_i)
5:          {(m_j^i, e_x, p)} ← filterCategories({(m_j^i, e_x, p)})
6:          addToLists(E_LIST, {(m_j^i, e_x, p)})
7:      while !stoppingCondition(Q^k, E_LIST) do
8:          m_j^i ← selectMention(E_LIST)
9:          e_x ← EL(m_j^i, t_i)
10:         updateLists(e_x, m_j^i, E_LIST)
11:     produceAnswer(Q^k, E_LIST)
```

3.1 Preparatory Phase

In order to prepare the mentions for the thinking and execution phase, The algorithm iteratevely loops over every tweet t_i in $\mathcal{T}$ and calls the LU function on t_i to return a list of mentions with their possible entities and the linking probabilities. After that, the entities are filtered based on the category of the query Q^k. Entities that are not associated with category c_g are removed and their linking probability are added to the OTHER probability for that mention.

The algorithm creates a list for every entity e_x associated with category c_g. The entity list (l_{e_x}) has an unresolved mention list which holds pointers to the mentions that could refer to entity e_x along with the linking probabilities. Mentions are sorted descendingly based on their linking probability allowing for instant access to mentions with highest & lowest probabilities if needed when selecting a mention to link. Every entity list l_{e_x} stores min & max counters ($min(l_{e_x})$ & $max(l_{e_x})$) corresponding to the number of linked mentions and the maximum number of mentions that could link to e_x respectively. Entity lists are held in another list and are sorted based on the max counter in a descending order as shown in Figure 2. Additionally, every mention is represented as an object where it has a list of pointers to all the possible entities it could refer to. The probability that mention m_j^i does not link to any possible entity is (1 - $\sum_{x=1}^{y} p(m_j^i, e_x)$, where e_x is an entity associated with category c_g) and this is considered the OTHER probability.

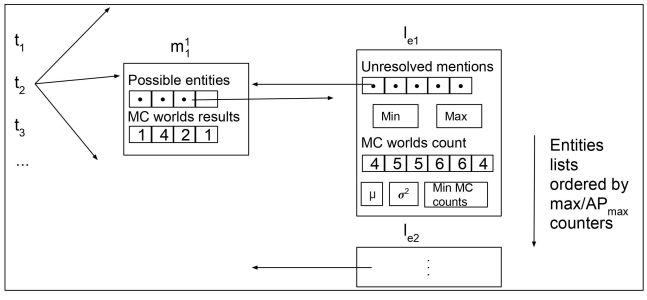

Figure 2: TQEL data structures.

3.2 Thinking & Execution Phase

Mention Selection. In order to save time in the query execution, we need to intelligently select the mentions to link that will help in reaching an answer the top-k query efficiently. The mention to be selected in the next iteration for dismabiguation is influenced by different factors (e.g. the current spot of the possible entities of m_j^i and the probability of linking mention m_j^i to entity e_x that is currently in the top-k). This process will vary when answering the query using the exact approach or the approximate approach and will be discussed later.

Entity Linking and Updating Entity Lists. If mention m_j^i is linked to entity e_x, l_{e_x} is updated accordingly. If mention m_j^i is shared among different possible entities, this will result in updating all the entity lists that have mention m_j^i in their unresolved mentions list as well. For example, in Figure 1, m_1^1 could refer to Black Panther (2018) with probability 0.6, Black Panther (1977) with probability 0.1 or OTHER with probability 0.3. After calling the EL function on m_1^1, we find that it refers to Black Panther (2018). When updating the lists, it is added to the list of Black Panther (2018) and deleted from the list of Black Panther (1977). After linking some of the mentions, the min & max of the affected entity lists will change causing the order of entity lists to be adjusted accordingly.

Stopping Condition. In this step, the algorithm checks if a solution for Q^k is found and can be validated. This test is different for the exact approach as well as the approximate approach and will be discussed in detail in Sections 4 & 5.

4 TQEL-EXACT APPROACH

To discuss TQEL-exact, we specify how to implement *stopping condition* and *mention selection* functions.

4.1 Stopping Condition

To check for the *stopping condition*, TQEL-exact maintains the minimum and maximum possible counts of mentions associated with each entity list. The *stopping condition* simply checks to see if there exists k entity lists that have min counters $\geq$ max counters of the rest of the entity lists. Since the entity lists are sorted in decreasing according to their max counters, we can check for the *stopping condition* by checking if the min values of the first k lists are greater than or equal to the max counter of $k+1$ entity list.

4.2 Mention Selection

Given entity lists sorted based on the max values, the *mention selection* strategy could either (a) choose mentions from lists with highest k max values to resolve that have a high probability of resolving to true in order to increase the min value of such lists, (b) choose mentions from the remaining lists with low probability of resolving to true in order to reduce the max value of those lists, or (c) choose mentions using a cost based strategy that minimizes the estimated cost of reaching the *stopping condition*.

Greedy Approach that greedily chooses the mention that has the highest linking probability from the entity list with the highest max counter. For example, in Figure 1, mention m_3^{11} will be chosen. After the mention is linked, the max and min counters of each list are updated along with the list order based to the result of entity linking. It will stop linking mentions from entity list e_i when $min(l_{e_i}) \geq max(l_{e_{k+1}})$ given that all the entity lists are sorted.

Benefit-based Approach that estimates the number of calls to entity linking needed (ENL) to prove that the k entities with the highest max counter is the top-k result set. It may either choose mentions from the first k entities with the highest max-values (with the hope this increases the min-values of those lists) or choose entities from the other lists (with the hope that it helps decrease their max-values). ENL is calculated as follows:

$$ENL = \sum_{i=1}^{k} maxof(max(l_{e_{k+1}}) - min(l_{e_i}),\ 0) \tag{2}$$

where maxof is a function that returns the max of two numbers.

Let $ENL(S)$ be the the expected number of calls to entity linking function for the given current entity lists and $ENL(S^{m_j^i,e_x})$ be expected number of calls to entity linking function when m_j^i links to e_x after calling the entity function on m_j^i. Then we choose the mention with the highest benefit function score (shown in Equation 3). If an entity list is guaranteed to be in the top-k answer set (i.e. it has a min counter $\geq max(l_{e_{k+1}})$), then mentions that are associated with such entity will not be considered for linking.

$$\begin{aligned} Benefit(m_j^i) = ENL(S) - (\sum_{m_j^i \in l_{e_x}} (p(m_j^i, e_x) \\ \times ENL(S^{m_j^i,e_x})) + p(m_j^i, OTHER) \times ENL(S^{m_j^i,OTHER})) \end{aligned} \tag{3}$$

The benefit-based approach can be optimized by limiting the calculation of the benefit function to be on mentions that indeed will reduce the value of ENL if linked. Such optimization can be achieved by calculating mentions from the first $2 \times k$ entities as they are the only mentions that will reduce ENL in the next iteration.

For example, using the benefit function for mentions in Figure 1, ENL(S) = 6 and mention m_3^{11} has a benefit score of: 6 - (0.74 * 5 + 0.08 * 5 + 0.18 * 6) = 0.82. On the other hand, mention m_2^6 has a benefit score of: 6 - (0.2 * 6 + 0.8 * 6) = 0. Therefore, choosing m_3^{11} is helpful, at this stage, while choosing m_2^6 is wasteful.

5 TQEL-APPROXIMATE APPROACH

We describe TQEL-approximate by specifying how we implement the three main functions in Algorithm 1, viz., functions for checking the *stopping condition*, *select mention*, and *update lists*. The latter two are straightforward once we have developed our strategy for checking the *stopping condition*. We, thus, focus the discussion to checking the *stopping condition* and will briefly describe the *select mention* and *update lists* functions at the end.

In TQEL-approximate, the *stopping condition* is weaker compared to that for the exact approach. In particular, TQEL-approximate stops early when it identifies k entities whose tkp is above a user specified threshold τ. TQEL-approximate iteratively calls the entity linking function to reduce uncertainty until the *stopping condition* is reached. This, in return, causes uncertainty in the counts associated with the entities to be reduced resulting in satisfying the *stopping condition* early.

Checking the *stopping condition* in TQEL-approximate, however, is complex. It requires the enumeration of all possible worlds based

on the linking probability of each mention and computing the probability of a particular entity to be in top-k. Since the number of such worlds is exponential, enumeration is not feasible. To address the above, we use an estimation based on sampling the possible worlds allowing us to check for the *stopping condition* (i.e., probability of all entities, in a result set, to be in the top-k is above τ) with high confidence, as discussed below. We note that while several works have explored top-k queries in probabilistic databases [9, 18, 34, 35, 39], as discussed in Section 7, none of the existing approaches have addressed the specific top-k query over count beyond using a sampling based approach we adopt.

5.1 Checking for the Stopping Condition

The *stopping condition* in TQEL-approximate uses Monte-Carlo (MC) simulation similar to prior work on probabilistic query processing [21], [28]. MC simulation relies on randomness to generate a sample world w_a from the set of possible worlds $\mathcal{W}$. To implement MC simulation, we iteratively select a mention m_j^i and assign an entity to that mention based on the linking probability distribution of all possible entities e_x that could be referred to by m_j^i (including the probability of assigning mention m_j^i to OTHER). After assigning all the mentions, a sample world w_a is generated and the result of the top-k query is computed. Given N sample worlds, we can compute the top-k answers to Q^k in those worlds, the results of which can be used to estimate the probability of an entity being in the top-k by using normal approximation to the binomial distribution. We compute $\hat{p_i}$ as in Equation 4 and use it to find the confidence interval for the entity's probability of being in the top-k.

$$\hat{p_i} = \frac{\text{number of times } e_x \text{ appears in top-k}}{N}$$
$$lcb = \hat{p_i} - z \times \sqrt{\frac{\hat{p_i} \times (1 - \hat{p_i})}{N}} \tag{4}$$

The z-score in the equation above is a critical value for defining the confidence interval and is decided based on the user-defined τ using a z-score table and the lower confidence bound (lcb), based on a normal approximation to the binomial distribution, accounts for the uncertainty due to the finite number of MC samples.

TQEL-approximate checks the *stopping condition* by checking lcb against τ. If lcb is above τ, the *stopping condition* has been reached and the top-k answer can be returned. However, if lcb is below τ, TQEL-approximate continues with linking additional mentions that changes the uncertainty of entities associated with the mention. With enough number of iterations, the *stopping condition* is guaranteed to be met (e.g., when all the mentions have been linked). The time complexity of the above implementation is is O(MN), where M is the number of mentions and N is the number of MC samples.

The above implementation of MC simulation to check the *stopping condition* suffers from two limitations. First, as specified, the approach has to pay the overhead of running an MC simulation (complexity order of $O(MN)$) after each EL call. One could batch multiple mentions to be linked into batches of size b to reduce overhead of running the MC simulation repeatedly after each EL call. The size of the batch b has to chosen carefully as sub-optimal choices could result in overheads. For instance, if b it is too large, we will pay the overhead of executing additional EL operations (which are also expensive) unnecessarily since calling the *stopping condition* without linking all the entities in the batch might also have met the required *stopping condition*. On the other hand, if we select b to be too small, we end up paying the overhead of the MC simulation repeatedly. A more efficient algorithm would minimize the number of EL calls performed prior to calling the *stopping condition* function while simultaneously guaranteeing that the *stopping condition* is met when it is called. Such an approach would minimize both the linking cost, as well as, the cost of MC simulations.

Another limitation is that each time the *stopping condition* is called, a new set of N samples are generated using the simulation. Such an overhead can be partially mitigated by observing that the samples obtained for different mentions assignments are independent and that between any two iterations (i.e., calls to the *stopping condition*) the probability values of only a small fraction of mentions (i.e., the mentions selected for linking in the previous batch) have changed. We can speed up the simulation by leveraging the work done during previous iterations. We next describe ways TQEL-approximate uses to overcome the above two limitations.

5.2 Exploiting Filters

To reduce the number of times we execute the expensive MC simulations, we employ a cheaper/less expensive filter to estimate if the *stopping condition* will be met by the MC simulations evaluation. Our motivation of invoking such a filter is analogous to the way blocking functions is used to reduce calls to the more expensive entity resolution function in the data cleaning literature [4, 37], or cheaper predicates (e.g., simple selections) are executed first prior to calling more expensive tests in query processing literature [6]. In our setting, if a filter fails (i.e., determines that the *stopping condition* will not be met by the MC simulation evaluation), we continue linking mentions to reduce uncertainly. This process proceeds until the filter condition is satisfied (i.e. determines that it is likely that the results will meet the stopping criteria after the MC simulations). If the filter succeeds the suggested results are fed into the MC simulation to check/validate if indeed the *stopping condition* has been reached. If not, then additional cleaning is performed and the filter is appropriately refined (i.e., made more conservative/tighter) in an iterative fashion. The filter based algorithm is depicted in Figure 3.

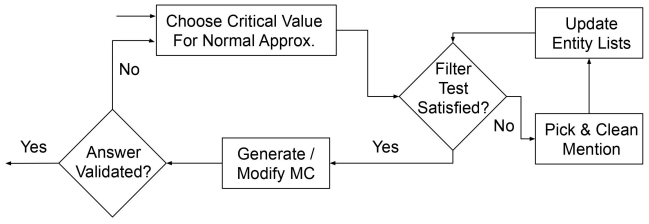

Figure 3: Stopping condition checking flow diagram

Effectively, our modified approach substitutes the more expensive MC simulation by a (much cheaper) filter until the algorithm reaches the point where we can be fairly confident that the *stopping condition* has been reached. To design an effective filter, we apply normal approximation statistics on the linking probabilities

of mentions associated with a specific entity e_x to compute the possible range of values for a given z-score (i.e., the z-value confidence interval, where z is a parameter associated with the filter). The confidence intervals can be computed efficiently while constructing the entity lists by computing the mean (μ) & standard deviation (σ) of e_x, as shown in Equation 5. We treat the upper & lower bounds of the confidence intervals (shown in Equation 7) as the approximate max & min (AP_{max} & AP_{min}).

$$\mu_{l_{e_x}} = \sum_{m_j^i \in l_{e_x}} p(m_j^i, e_x) \tag{5}$$

$$\sigma^2_{l_{e_x}} = \sum_{m_j^i \in l_{e_x}} p(m_j^i, e_x) \times (1 - p(m_j^i, e_x)) \tag{6}$$

$$AP_{min} = \mu - x\sigma, \qquad AP_{max} = \mu + x\sigma \tag{7}$$

where x is the z-score associated with the desired confidence level. For example, for the confidence level of 95%, $x = 1.96$, etc.

From the perspective of efficiency, the filter based on normal distribution assumption, takes roughly 1.7 *ms* to calculate the AP_{max} & AP_{min} for 10000 entities. In contrast, MC takes about 3.13 *ms* to generate only 1 world sample for 10000 mentions, which for a sample size of say a 1000 would take approximately 3 seconds.

After calculating the AP_{max} & AP_{min} for all entity lists, the filter checks if a solution is reached based on its calculated approximation. Given that we sort the entity lists descendingly based on their AP_{max} values, if there exists k entity lists with $AP_{min} \geq AP_{max}$ of the entity that is in the $k+1$ position then the condition is met and the suggested result is validated using the MC simulations.

5.2.1 Choosing the Confidence Intervals. Note that the choice of x in Equation 7 does not influence the correctness of the approach since, in our case, the filter is only used as a hint, and we still perform the MC simulation for validation of the top-k answer. However, the value x plays an important role in determining the number of calls to entity linking functions and the cost of generating and updating the MC simulation results. Smaller the value of x, smaller the confidence interval, and, lesser the number of calls to entity linking function in order to meet the filter condition (i.e., there exists k entities such that their $AP_{min} \geq AP_{max}$ of the rest of the possible entities). Conversely, larger the value of x, more entity linking functions will need to be called before the filter condition is met. Thus, choice of x plays the key role in determining the number of entity linking functions called and the number of times MC simulation is invoked. An optimal value $x*$ would allow for enough entity linking functions to be called such that the MC simulations test succeeds when called for validation.

To find such an $x*$, we first introduce an estimation of the number of calls to the entity linking function in order to find a candidate top-k answer (ENC(α_x)) given that x is the z-score for the confidence α_x. Given uncertainty of linking function, estimating such a number is complex. We use a heuristic estimation that greedily favours mentions that are associated with the k entities having the highest AP_{max} and estimate the number of mentions to be cleaned on those lists in order to report them as a candidate top-k result. The estimation is as follows:

$$ENC(\alpha_x) = \sum_{i=1}^{k} \frac{maxof(AP_{max}(l_{e_{k+1}}, x) - AP_{min}(l_{e_i}, x),\ 0)}{1 - \frac{\mu_{e_i}}{max(e_i)}} \tag{8}$$

Where $AP_{min}(l_{e_i}, x)$ is the $AP_{min}(l_{e_i})$ if x is the z-value.

$ENC(\alpha_x)$ estimates the number of EL calls needed to shift the μ of the first k entities, given they are sorted on their AP_{max} values, so that their $AP_{min} \geq AP_{max}(l_{e_{k+1}})$. Note that $ENC(\alpha_x)$ is monotonic in x, i.e. as x value increases ENC(α_x) also increases. To see this note that $AP_{max}(l_{e_i}, x)$ increases and $AP_{min}(l_{e_i}, x)$ decreases as value of x increases. Thus, the numerator in Equation 8, increases with the increase in x while the denominator is a constant. Thus the function is monotonic.

We, further, need to quantify the probability of success and failure if an arbitrary x value is chosen. Using normal approximation, let us say there are two entity lists l_{e_i} & l_{e_j} such that $AP_{min}(l_{e_i}) \leq AP_{max}(l_{e_j})$. The probability that l_{e_i} count is below $AP_{min}(l_{e_i}) = (\frac{1-\alpha}{2})$ if the mentions' probabilities follow normal distribution. Similarly the probability of the count of l_{e_j} to be above $AP_{max}(l_{e_j})$ is the same. We conservatively assume that when count of l_{e_i} is below $AP_{min}(l_{e_i})$ or the count of l_{e_j} is greater than $AP_{max}(l_{e_j})$, the filter test ($AP_{min}(l_{e_i})$ is $\geq AP_{max}(l_{e_j})$ would fail during validation. We, thus, estimate the probability of success at the validation stage, given the success of the filter, to be: $(1 - 2 \times \frac{1-\alpha}{2}) = \alpha$. Likewise, the probability of the validation failing given the success of the filter is (1- α).

We now calculate the cost of TQEL-approximate given an x value as $cost(\alpha_x)$, in the case that the validation step succeeds. Let C_{EL} be the cost of the entity linking function, C_{MC} be the cost of generating N MC simulations for one mention and C_V as the cost of running top-k query on all N worlds to verify the answer. $cost(\alpha_x)$ can be calculated as follows:

$$cost(\alpha_x) = C_{EL} \times ENC(\alpha_x) + C_{MC}(M - ENC(\alpha_x)) + C_V \tag{9}$$

We next consider the cost of TQEL-approximate for a given x value if the validation step fails. In such a case, we need to clean more mentions to get better results. We do that by using a higher x value such that the confidence that the suggested top-k result succeeds is higher. Such a scenario requires paying an overhead of linking more mentions, updating their linking results in the stored MC simulations and executing the verification process again. Consider that we begin TQEL-approximate with a value of $x = x_1$, that results in failure at validation which prompts the algorithm to use $x = x_2$, which succeeds. We denote the cost of such an execution by $cost([\alpha_{x_1}, \alpha_{x_2}])$. We estimate C($\alpha_{x_1}, \alpha_{x_2}$), which is the cost of choosing a value x_1 where the suggested top-k answer set based on x_1 estimation fails and we choose a higher value x_2, as follows:

$$\begin{aligned} cost([\alpha_{x_1}, \alpha_{x_2}]) &= cost(\alpha_{x_1}) \\ &+ (C_{EL} + C_{MC})(ENC(\alpha_{x_2}) - ENC(\alpha_{x_1})) + C_V \end{aligned} \tag{10}$$

In the equation above, note that the case when $\alpha_{x_2} = 1$ (i.e., the confidence interval covers the entire distribution) is special. In such a case, to meet the filter condition, one will need to link all ambiguous entities and there will be no uncertainty in the top-k results. Thus, the process will not need to execute the final validation step. In such a case, the cost would be:

$$cost([\alpha_{x_1}, 1]) = cost(\alpha_{x_1}) + C_{EL}(M - ENC(\alpha_{x_1})) \tag{11}$$

Algorithm 2 Choosing the critical value

```
1: procedure ChoosingCriticalValue(M, C_EL, C_MC, C_V, τ, budget)   α_x ← τ, α_x' ← 1, b ← 0
2:     while α_x < α_x' AND b < budget do
3:         if EC([α_x, α_x']) ≤ EC(α_x') then
4:             α_x' ← (α_x' + α_x)/2, b ← b + 1
5:         else
6:             α_x ← α_x', α_x' ← 1, b ← 0
7:     return α_x
8:     =0
```

We next introduce a general equation for estimating the expected cost ($EC([\alpha_{x_1}, \alpha_{x_2}, ..., \alpha_{x_n}])$) as follows:

$$EC([\alpha_{x_1}, \alpha_{x_2}, ..., \alpha_{x_n}]) = \alpha_{x_1} cost(\alpha_{x_1}) + (\alpha_{x_2} - \alpha_{x_1}) cost([\alpha_{x_1}, \alpha_{x_2}]) + ... + (\alpha_{x_{n-1}} - \alpha_{x_n}) cost([\alpha_{x_1}, \alpha_{x_2}, ...\alpha_{x_n}]) + (1 - \alpha_{x_n}) cost([\alpha_{x_1}, ..., \alpha_{x_n}, 1) \quad (12)$$

TQEL-approximate uses Algorithm 2 to find the optimal value of x. The algorithm searches for an x value such that $\text{EC}(\alpha_x) \leq \text{EC}(\alpha_{x'})$ for any $x' \geq x$. The algorithm chooses an initial x value such that α_x = user-defined τ. The intuition is that we want to choose an x such that the probability of success when validating top-k results given the success of the filter ≥ user-defined τ. If we choose x < τ, even if the filter succeeds the chance that the validation using MC simulation will fail is high. Given the monotonic nature of $\text{ENC}(\alpha_x)$, The algorithm performs a binary search to find a x' value such that $\text{EC}(\alpha_{x'}) < \text{EC}(\alpha_x)$. If no such value x' is found before a searching budget b is exhausted, we choose x as the critical value. However, if an x' value is found before b is exhausted, we choose x' as the new value for x and we rerun the algorithm.

5.2.2 Efficacy of Filter. The filter provides a cheaper mechanism to estimate the outcome of the query without having to pay for the overhead of the real test. While TQEL's correctness is independent of the filter, its effectiveness in reducing overhead of unnecessary EL calls and expensive MC based verification depend upon how effective the filter is. Therefore, we define a metric to measure how well the filter performs on different queries, datasets and probabilities distribution of the top-k entity lists.

The filter in TQEL may determine the *stopping condition* has been reached or not. For each such case, either the MC simulation would agree with the filter's determination or would disagree resulting in 4 possible cases corresponding to true positive (TP) when both the filter and MC simulator determine that the *stopping condition* has been reached, false positive (FP) when the filer determines the *stopping condition* has been reached but the MC does not, true negative (TN), when both agree the *stopping condition* is not reached, and false negative (FN) when the filter does not determine that *stopping condition* has reached, but MC, if it is executed would have determined that the condition has been reached. Note that FP would result in an unnecessary call to the expensive verification, and the FN would result in an unnecessary call to the EL function.

In TQEL, TP is always 1 and FP corresponds to number of times the filter test is satisfied but the validation using MC simulation is not. FN can be calculated by subtracting (the necessary EL calls) from the actual number of calls performed when using the filter, while TP refers to the number of times the filter test accurately predicted the negative outcome. Given the above, we can measure the quality of the filter using any of the metrics one uses for measuring performance of binary classifiers (e.g., precision, recall, specificity, miss rate, etc) [33, 36]. We choose accuracy defined as the ratio of the sum of TP and TF divided by the total sum of TP, TN, FP and FN since both false positives and negatives result in expensive calls to MC simulation / EL function respectively. We report in experiment 7, a detailed accuracy evaluation.

5.3 Monte-Carlo Simulation Implementation

In order to efficiently access the previous MC simulations, we keep the results of the previous runs in a vector of size N inside the mention object. For each value of the vector $v_1, v_2, ..., v_N$, we store the entity that is assigned to that mention for that specific sampled world. We, further, store the number of occurrences of each entity e_x in the N sampled worlds in a vector of size N where each value v_a corresponds to the number of assigned mentions for e_x in world w_a as shown in Figure 2. We store such values to easily retrieve the number of occurrences for that entity in each sampled world w_a. We also store the minimum number of occurrences of all N worlds for each entity list.

In order to smartly generate MC samples for mentions in TQEL-approximate, we start by generating the samples for mentions of entity lists with the highest AP_{max} value. We iterate over the entity lists in a decreasing order of AP_{max} values in order to generate samples for the mentions which are associated with entities that are competing to be in the top-k answer. If k entity lists were found such that their minimum number of occurrences ≥ $max(e_j)$ such that e_j is not an entity from the first k entities, we do not perform MC simulation process for e_j.

5.3.1 Answer Verification. We use Equation 4 to calculate the lcb for the first k entities given that they are sorted based on their AP_{max} values. If for every entity, $lcb > \tau$, we return the first k entities as the answer to Q^k. However, if this step fails, we continue to link more mentions until the *stopping condition* is met.

5.4 Mention Selection

TQEL-approximate follows a benefit-based function approach similar to Equation 3 where ENL is replaced by ENC. We define $ENC(S)$ as the expected number of calls such that the filter test passes given the current entity lists and $ENC(S^{m_j^i, e_x})$ as the expected number of calls needed for the filter to pass when m_j^i links to e_x. The benefit function finds a mention m_j^i such that if linked brings us closer to the top-k answer result. Note that choosing the entity with the highest linking probability, say 0.9, might not always be the best answer since the reduction in uncertainty, if it links to the desired entity, is quite low. Therefore, the benefit function takes into account the probability of success as well as the reduction in the expected number of EL calls.

5.5 Updating Lists and Approximations

5.5.1 AP_{min} and AP_{max} Maintenance. We can instantly calculate and update the AP_{min} & AP_{max} values based on the stored values of μ & σ for every entity list. Values of μ & σ are updated by

adding the probability value of new mention m_j^i to μ and adding p(m_j^i)) * 1 - p(m_j^i)) to σ^2. Moreover, whenever we link mention m_j^i to an entity e_x, for every entity that was associated with mention m_j^i we update the values of μ & σ accordingly. To update entity e_x which is linked to mention m_j^i we add (1 - p(m_j^i)) to μ and subtract p(m_j^i)) * 1 - p(m_j^i)) from σ^2. For the other entities we subtract p(m_j^i)) from μ and subtract p(m_j^i)) * 1 - p(m_j^i)) from σ^2 as well.

5.5.2 Monte-Carlo Simulation Maintenance. In order to properly maintain the samples of the Monte-Carlo simulation algorithm, we store the results of the previous N runs to use ,if needed, in later stages. Whenever a mention m_j^i is linked to entity e_x, we update the number of occurrences of all the entities that could be referred to by m_j^i. We remove m_j^i from any entity that is not e_x and add it to e_x in all the runs too. If the mention m_j^i is not linked to any entity, then we remove m_j^i in all N runs of the possible entities.

6 EXPERIMENTS

6.1 Experimental Setup

Datasets. We used two tweets datasets that have been collected from Twitter's public API without any specification or focus on certain keywords, locations or topics. All these tweets are English tweets. We ran the two heuristics of TQEL-exact & TQEL-approximate on both datasets and have conducted multiple experiments to evaluate such approaches with different settings. We have also used a synthetic dataset that have been generated by introducing a 10% uniform noise to the linking probability of every mention-entity pair using the small dataset. This was done by multiplying a randomly generated real number between (0.9, 1.1) to every mentions linking probabilities. Afterwards, we normalize the probabilities in order to satisfy that the sum of probabilities equals 1. Our goal is to test the robustness of TQEL when the initial linking probability of a mention-entity pair is erroneous and study how TQEL-exact and TQEL-approximate perform in such circumstances. For the TQEL-approximate we have chosen 10,000 to be the number of runs for the MC simulation for all of our experiments.

• **Small Dataset.** The first dataset contains 101,486 tweets and has been collected from April 6-April 7, 2018.

• **Large Dataset.** The second dataset contains 11,250,894 tweets and has been collected from May 30-June 9, 2019.

Approaches: In our experiments we use 6 baselines to compare them with TQEL-approximate approach and they are as follows:

• **Random Approach (random).** In this approach we iteratively choose a random mention to link until a solution is found. We enhance the efficiency of this algorithm by limiting the choice of mentions from lists that are competing to be in the top-k answer.

• **TQEL-exact (Benefit Function) Approach.** Discussed in 4.2.

• **TQEL-exact (Greedy) Approach.** Discussed in 4.2.

• **TQEL-approximate (Greedy) Approach.** For this approach, TQEL chooses the mention with the highest probability in the list rather than using the mention selection technique discussed in 5.4.

• **NOFILTER Approach.** In this approach, TQEL does not use the filter proposed in 5.2 but rather validates the query using MC Simulation after each EL call. This approach follows the same mention selection technique for TQEL-approximate.

• **MC-NOOPT Approach.** In this approach, TQEL does not apply the optimizations discussed in 5.3 & 5.5 but rather runs the Monte-Carlo simulation every time a validation is needed.

Knowledge Base: We have used Wikipedia [3] as the source of our KB in this experiment by indexing the whole Wikipedia dump using Apache Lucene [1] to make it easier to query titles, text bodies and other metadata. We have also tagged each Wikipedia article with categories fetched from DBpedia to be able to answer the top-k query with the category filter. We have only indexed the articles that are included in our queries and the indexing process took around a day and 20 hours.

Queries: the queries that are used for the experiments are top-k queries of different categories in DBpedia. We have used multiple categories from different levels in the DBpedia category hierarchy in order to control the selectivity of the query. The selectivity is corresponding to the number of mentions in all the tweets. For example, a 10% selectivity indicates that 90% of the mentions will be discarded as they do not have any possible entity that is associated with the category c_g. We also execute the query on different k-values and confidence scores. The categories that are used in the experiments have been selected from 3 different levels in the category hierarchy and are as follows:

• **Top-level categories:** Agent, Work and Place. With an average selectivity of 55%.

• **Med-level categories:** Person, Musicalwork and Organization. The average selectivity is 31%.

• **Low-level categories:** Film, Song, Populatedplace, Artist, Athlete and Politician. The average selectivity is 13%.

Entity Extraction & Lookup Function: In our experiment, we use simple dictionary-based entity lookup function that parses the tweet sequentially and identifies the largest sequence of words that matches an article in the KB. When there is a match, LU generates the possible candidates for the identified mention and give each candidate a linking probability based on a score of different factors (e.g. page view count). LU takes roughly 0.5 milliseconds per tweet.

Entity Linking Function: In order to disambiguate a mention-entity pair we use TagMe API [13] that returns an entity for the mention in the tweet text along with a probability of linking. In our setting we use entity linking function as a determining function by assigning such mention to the entity if the linking function returns a probability of 0.5 or more and vice versa. The linking process takes on average 44 milliseconds.

6.2 Experiments results

Experiment 1: Number of entity linking calls & execution time for different K values. In this experiment we show the effectiveness when using different strategies and the advantage that the TQEL-approximate approach provides as a function of k. We average the query results based on the category used over the hierarchy levels. Total execution time for TQEL-exact consists of thinking time, benefit function calculation, and the time needed to resolve the chosen mentions. Moreover, for TQEL-approximate total execution time is calculated by adding critical factor choosing time, mention selection time, time to run MC simulation and time to validate the top-k result using the MC runs.

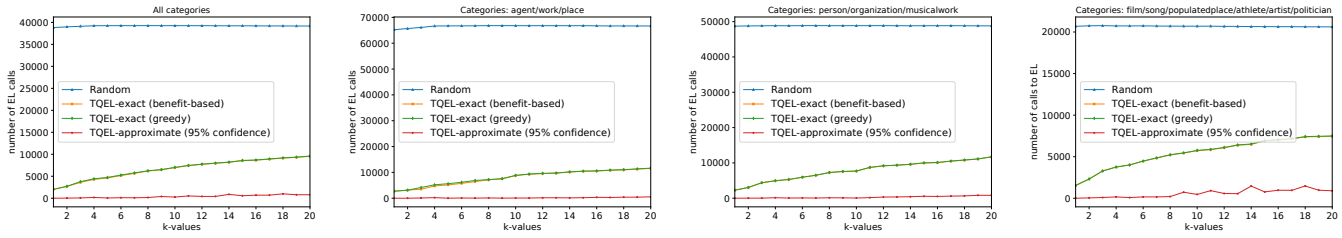

Figure 4: Comparing number of calls to entity linking function vs different k-values for multiple categories

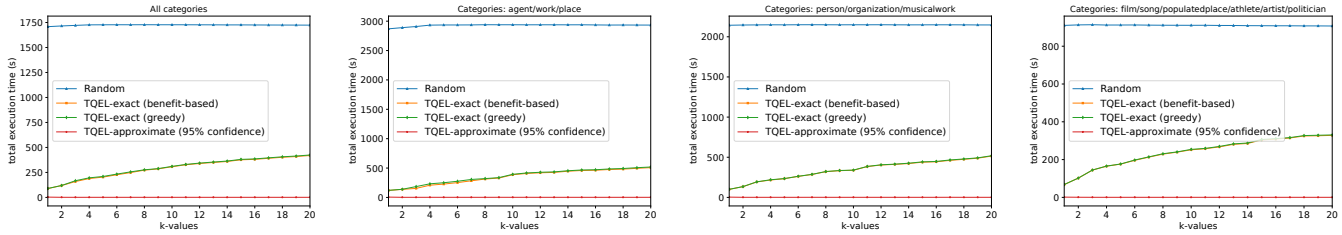

Figure 5: Comparing total execution time (seconds) vs different k-values for multiple categories

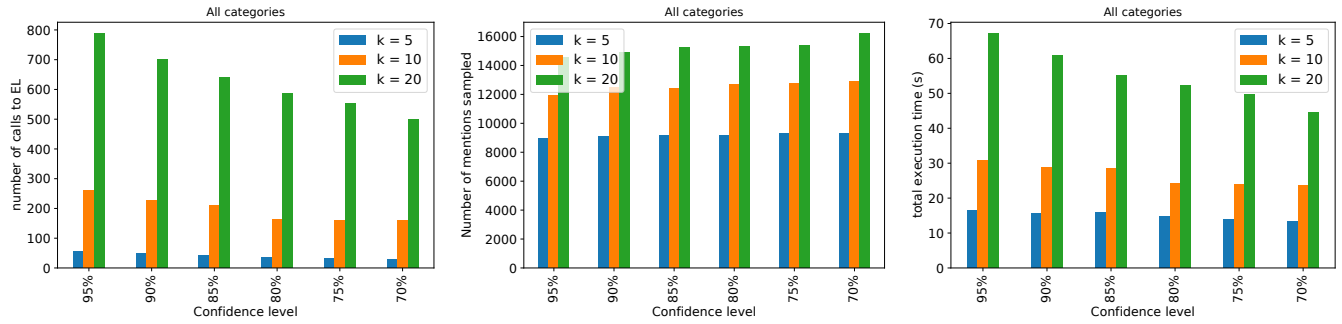

Figure 6: Detailed performance analysis of TQEL-approximate using different confidence levels.

From Figures 4 & 5, we can see that TQEL-approximate is showing promising results in terms of total of number of calls to the entity linking function and total query execution time. We noticed that in all the queries we are saving in term of calls to EL function, which is a bottleneck for the query, due to the use of approximate query answering techniques. This is due to the fact that the variance in the number of occurrences of each entity in the top-k is quite large and in order to find a top-k answer we only perform a much smaller number of EL calls.

We also see another trend in TQEL-approximate approach where the number of EL calls for $k = x$ is higher compared to $k = x-1$ and $k = x+1$, this happens because of having two or more contenders that have relatively comparable counts which in return forces more calls to EL function for mentions in the competing entities.

We illustrate the fact that TQEL-exact (for both heuristics) outperforms the random approach in general by a huge margin (130x saving) by smartly selecting mentions to disambiguate in an iterative fashion. The reason behind that is by focusing on proving that the k-highest entities in terms of max values. We also see the number of calls and total execution time is increasing whenever k increases while in TQEL-approximate that is not the case. The reason behind such results is that, if the approximation clearly shows that the top-k result without any competition from the $k+1$ entities, then little work in terms of entity linking is needed regardless of k. However, for TQEL-exact, we need to pay the overhead of performing such EL calls to make sure that the top-k result is valid.

In the experiments' figures, we do not include the execution time of the entity extraction and lookup function since it is shared by all strategies. Additionally, this process could be executed on the entire dataset during ingestion time since it is cheap. We also do not include the numbers of the naive approach that requires cleaning all tuples before running the query although it is clear that our proposed strategies outperform such strategy. The naive approach takes around 145,023 EL calls and around 1.8 hours to execute. We do not report the naive approach numbers as it is the same over all queries and stretches the figures making them less readable.

Experiment 2: Detailed analysis of TQEL-approximate performance. In this experiment we analyze the performance of TQEL-approximate for different confidence levels that are given by the query. We will focus on 3 factors that are of interest: number of EL calls, number of mentions sampled using our MC technique and total execution time of the query. We also report the average of the query results over all categories for this experiment.

In the left sub-figure of Figure 6, we can see that the confidence level heavily affects the number of entity linking calls that ends up being executed. The number of calls to EL function is growing as the confidence level rises. The number of calls to EL is also affected by the k value as it might cause a significant increase in EL calls due to the competitiveness of entities for that rank based on the number of occurrences.

We also report the difference in number of mentions required for the MC simulation in order to generate a top-k answer set. The middle sub-figure in Figure 6 illustrates the amount of savings that are achieved by smartly limiting the generation of MC simulations to mentions associated with competing entities as discussed in 5.3. The number of sampled mentions also decreases when the number of resolved mentions increases as we will not need to generate samples for the linked mentions.

In the last sub-figure of Figure 6, we report the total execution time of the queries which consists of (critical factor choosing time, mention selection time, time to run MC simulation and validate top-k answer set using the MC runs). In this experiment we illustrate the fact that in general, choosing a lower τ leads to less number of entity linking calls and therefore less overall execution time due to the fact of less EL calls.

Experiment 3: Scalability of TQEL. This experiment illustrates the impact of large datasets on TQEL-exact & TQEL-approximate and that the amount of savings we are able to achieve using the TQEL-approximate is noticeable. We have run the query on the category "film" and reported the performance over multiple k values. The "film" category had roughly 10% selectivity in the large dataset. From Figure 7, we see the same pattern or trend where the savings by TQEL-exact approach in term of EL calls is more than 100x for the calls needed to fully disambiguate all the mentions in the tweets. TQEL-approximate results are also promising and are quite similar to results that have been achieved on smaller dataset, even though in the large dataset we are dealing with a large number of tweets. This is due to the fact that the difference of number of occurrences between entities that are in the top-k is quite large which is allowing TQEL-approximate to return the result with high confidence without much entity linking calls. This experiment clearly shows that in real-world datasets differences in the number of occurrences should be exploited by returning an answer with high confidence using approximation techniques rather than paying the huge overhead of reporting the exact answer.

Experiment 4: Score of TQEL-approximate results. In this experiment we measure the accuracy for the results returned by TQEL-approximate for different τ values. To evaluate the returned answer set we have used two metrics:

- **Precision:** Represents the fraction of elements in the approximate answer set compared to the exact top-k set.
- **Rank Distance:** A modified version of the footrule distance to compute the distance of the inaccurate entity e_i in the approximate answer set to their exact rank. To compute rank distance we compute the following: $\frac{1}{k}\sum_{i=1}^{k} maxof(exact_{e_i} - k, 0)$ where $exact_{e_i}$ is the exact rank of e_i. We modified the distance calculation since we only report the top-k answer set without any order between the top-k answer set.

In Table 2, we averaged the scores of each confidence levels over all categories and over k values (1 - 20). We see that rank distance is heavily influenced by the chosen τ and the difference in EL calls.

Experiment 5: Robustness of TQEL. this experiment illustrates the effect of introducing noise to the linking probability to study its impact on the query execution time and answer quality.

Table 2: Evaluation metrics for different confidence levels

Confidence	95%	90%	85%	80%	75%	70%
Precision	0.95	0.93	0.92	0.92	0.91	0.90
Rank Distance	0.069	0.091	0.113	0.130	0.160	0.168

From Figure 8, TQEL-approximate approach is quite resilient and dominates other approaches in terms of the number of EL calls. However, the answer quality has dropped where the precision is 0.90 and rank distance is 0.089 for the confidence of 95%.

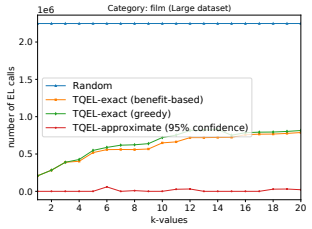

Figure 7: EL calls vs k-values for large dataset.

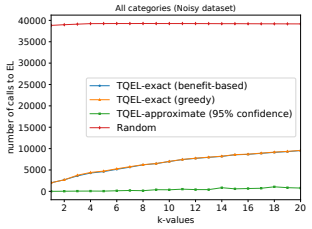

Figure 8: EL calls vs k-values for noisy dataset.

Experiment 6: Comparing TQEL with other approximate baselines. In this experiment we report the total execution time of the query for different proposed baselines. In Figure 9, we illustrate the execution time of the EL functions, denoted as [baseline] EL, in a different color to show how different baselines perform. NOFILTER approach was as expected the best in terms of EL function time since we validate the query after each call resulting in not executing any unnecessary EL calls. However, in terms of overall execution of the query it performed the worst due to the extremely high number of expensive verification overhead (factor of number of EL calls). We stopped running the query after 5 minutes due to the overhead caused by some baselines. MC-NOOPT approach verification cost was also high compared to TQEL-approximate since MC simulations will be run each time the validation step is performed. This clearly shows the effectiveness of our proposed MC optimizations. We see that TQEL-approximate performed better than TQEL-approximate (Greedy), especially with the execution time of the EL functions caused by the 9% increase of EL calls compared to TQEL-approximate.

Experiment 7: Evaluating Filter's Efficacy. This experiment shows the accuracy of TQEL-approximate, TQEL-approximate (Greedy) and NOFILTER baselines. In order to calculate the accuracy for the first two, we follow the measurement discussed in Section 5.2.2. In the case of NOFILTER, since it never characterizes data as negative (it always invokes the expensive MC simulation), the values for TN and FN are 0 and the value of TP is 1, while the value of FP are number of EL calls - 1. As a result, its accuracy is very poor (below 1% for all categories) we do not show the results in Figure 10. Figure 10 illustrates the accuracy of TQEL-approximate & TQEL-approximate (Greedy) with a reported average of 83% and 64% accuracy, respectively, over all categories. In the experiment we further analyzed and realized a correlation between the distribution of the top-k entity lists mentions' probabilities and the accuracy of the result. For the `agent` category where the size of the top-20 entity lists is relatively large, we get an accuracy of 95%. For agent, the average size of the top-20 lists was 149.75, the average σ based on normal approximation was 4.68 and the average μ difference between normal approximation and MC simulation was 0.1% (thus normal approximation is a better estimate), we get an accuracy of 95%. On the other hand, the same numbers for the `athlete` category were 47.56, 1.7 and 4.1% which effects the normal distribution statistics to be inaccurate resulting in a lower filter accuracy.

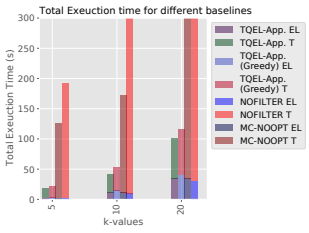

Figure 9: Execution Time

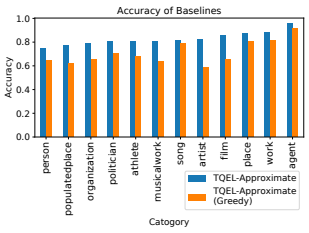

Figure 10: Accuracy

7 RELATED WORKS

Probabilistic Top-k Evaluation. Based on the survey [20], the top-k queries in probabilistic databases can be characterized into top-k selection queries, top-k join queries and top-k aggregation queries. Furthermore, top-k queries are evaluated under different query semantics: U-Topk, U-kRanks [34, 35], Global-Topk [39] and PT-k [18]. The query in TQEL corresponds to a top-k aggregation query over count using a slightly modified version of PT-k [18] semantics. As discussed clearly in [35], approaches to evaluate top-k selection queries [20] cannot be applied to evaluate top-k aggregation queries. Prior work on top-k aggregation [35] considers aggregate functions such as AVERAGE and SUM. While their algorithm would work with top-k COUNT queries we study, it would result in an exponential blowup making it hard to execute the top-k query (we have implemented their strategy and ran out of memory even for a small dataset when executing our query). Moreover, as stated in [35], their approach incurs extra overhead and might not be an efficient way for solving top-k aggregation problems that follow our answer semantics. The algorithm based on MC sampling we use, while simple, is likely the most practical approach to implementing top-k aggregate count queries. Note that the specific choice of the top-k evaluation algorithm is not our major contribution but we rather focus on reducing the overhead of entity linking using the query context.

Query-Time Data Cleaning. There have been emerging body of work that studied problems related to query-time data cleaning [4, 15, 15, 37] that focus on reducing the number cleaning steps in order to efficiently answer queries and treat data cleaning algorithms as black boxes. However, in [4, 15] they do not focus on ranking queries such as top-k queries. Cleaning dirty data to answer top-k aggregation queries have been studied previously in [37] in the context of the entity resolution problem. They proposed an interactive algorithm for Locality-Sensitive Hashing that essentially exploits the top-k semantics to cut down the cost of blocking only to clusters that are dense so that the blocking cost per tuple on sparse areas of data are very small. Our approach, similar to this work, also exploits semantics of the top-k operator to reduce costs. However, the context and solutions are different. While their work focused on top-k with clustering/grouping, our work exploits semantics in the context of classification/lookup type of entity disambiguation in the terminology of [7]. While the intuition is shared, the techniques are entirely different. In addition, in our settings, simply exploiting top-k semantics deterministically does not offer enough speedup, and thus, the main contribution of this paper is an approximate version which the prior work did not explore. [7] considers the problem of cleaning for top-k query in uncertain databases. However, it only considered top-k selection queries and hence not comparable to TQEL. Top-k queries in the context of crowdsourcing [8, 11, 23, 25, 26] have also tried to minimize the crowdsourced cleaning work. The model, however, is very different from what we use in TQEL - e.g., in such work a human worker, given 2 or more lists, provides ordering amongst the lists. Such is not the case by calling a linking function in TQEL making their solution inapplicable to our setting.

8 CONCLUSION

In this paper, we presented TQEL, a framework to support queries that retrieve top-k entities belonging to a user-specified category in a collection $\mathcal{T}$. TQEL exploits the query semantics to reduce the number of entity linking function calls for mentions in $\mathcal{T}$. It provides the option of answering the query exactly with deterministic guarantees or approximately with probabilistic guarantees. The paper focuses on the approximate approach that uses an implementation of Monte-Carlo technique to efficiently evaluate the query with guarantees. To reduce the overhead of Monte-Carlo simulations (which is expensive), TQEL-approximate uses normal approximation to estimate the count of each entity as a filter. To reduce the number of calls to the entity linking function, a benefit-based function is used to select mentions to link. In future work, we aim to explore the possibility of using such a filter-based approach for different queries that require data cleaning (e.g. SPJ queries).

ACKNOWLEDGMENTS

This work was supported by NSF Grants 1527536, 1545071, 2032525, 1952247, 1528995, 2008993, 2044107, 2139103 and DARPA under Agreement No. FA8750-16-2- 0021. Abdulrahman Alsaudi was supported by KSU's Graduate Studies Scholarship.

REFERENCES

[1] 2021. Apache Lucene. https://lucene.apache.org/.

[2] 2021. Wikipedia. https://www.wikipedia.org.

[3] 2021. Wikipedia:Database download. https://en.wikipedia.org/wiki/Wikipedia:Database_download.

[4] Hotham Altwaijry, Sharad Mehrotra, and Dmitri V Kalashnikov. 2015. Query: A framework for integrating entity resolution with query processing. *Proceedings of the VLDB Endowment* 9, 3 (2015), 120–131.

[5] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*. Springer, 722–735.

[6] Ron Avnur and Joseph M Hellerstein. 2000. Eddies: Continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 261–272.

[7] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. 2003. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 313–324.

[8] Eleonora Ciceri, Piero Fraternali, Davide Martinenghi, and Marco Tagliasacchi. 2015. Crowdsourcing for top-k query processing over uncertain data. *IEEE Transactions on Knowledge and Data Engineering* 28, 1 (2015), 41–53.

[9] Graham Cormode, Feifei Li, and Ke Yi. 2009. Semantics of ranking queries for probabilistic data and expected ranks. In *2009 IEEE 25th International Conference on Data Engineering*. IEEE, 305–316.

[10] Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudré-Mauroux. 2012. ZenCrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *Proceedings of the 21st international conference on World Wide Web*. 469–478.

[11] Eyal Dushkin and Tova Milo. 2018. Top-k sorting under partial order information. In *Proceedings of the 2018 International Conference on Management of Data*. 1007–1019.

[12] MS Fabian, K Gjergji, WEIKUM Gerhard, et al. 2007. Yago: A core of semantic knowledge unifying wordnet and wikipedia. In *16th International World Wide Web Conference, WWW*. 697–706.

[13] Paolo Ferragina and Ugo Scaiella. 2010. Tagme: on-the-fly annotation of short text fragments (by wikipedia entities). In *Proceedings of the 19th ACM international conference on Information and knowledge management*. 1625–1628.

[14] Abhishek Gattani, Digvijay S Lamba, Nikesh Garera, Mitul Tiwari, Xiaoyong Chai, Sanjib Das, Sri Subramaniam, Anand Rajaraman, Venky Harinarayan, and AnHai Doan. 2013. Entity extraction, linking, classification, and tagging for social media: a wikipedia-based approach. *Proceedings of the VLDB Endowment* 6, 11 (2013), 1126–1137.

[15] Stella Giannakopoulou, Manos Karpathiotakis, and Anastasia Ailamaki. 2020. Cleaning Denial Constraint Violations through Relaxation. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 805–815.

[16] Stephen Guo, Ming-Wei Chang, and Emre Kiciman. 2013. To link or not to link? a study on end-to-end tweet entity linking. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 1020–1030.

[17] Xianpei Han and Jun Zhao. 2009. NLPR_KBP in TAC 2009 KBP Track: A Two-Stage Method to Entity Linking.. In *TAC*. Citeseer.

[18] Ming Hua, Jian Pei, Wenjie Zhang, and Xuemin Lin. 2008. Ranking queries on uncertain data: a probabilistic threshold approach. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 673–686.

[19] Wen Hua, Kai Zheng, and Xiaofang Zhou. 2015. Microblog entity linking with social temporal context. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 1761–1775.

[20] Ihab F Ilyas, George Beskales, and Mohamed A Soliman. 2008. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)* 40, 4 (2008), 1–58.

[21] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher Jermaine, and Peter J Haas. 2008. MCDB: a monte carlo approach to managing uncertain data. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 687–700.

[22] Heng Ji, Ralph Grishman, Hoa Trang Dang, Kira Griffitt, and Joe Ellis. 2010. Overview of the TAC 2010 knowledge base population track. In *Third text analysis conference (TAC 2010)*, Vol. 3. 3–3.

[23] Jongwuk Lee, Dongwon Lee, and Seung-won Hwang. 2017. CrowdK: Answering top-k queries with crowdsourcing. *Information Sciences* 399 (2017), 98–120.

[24] Rui Li, Shengjie Wang, and Kevin Chen-Chuan Chang. 2013. Towards social data platform: Automatic topic-focused monitor for twitter stream. *Proceedings of the VLDB Endowment* 6, 14 (2013), 1966–1977.

[25] Yan Li, Hao Wang, Ngai Meng Kou, Zhiguo Gong, et al. 2020. Crowdsourced top-k queries by pairwise preference judgments with confidence and budget control. *The VLDB Journal* (2020), 1–25.

[26] Xin Lin, Jianliang Xu, Haibo Hu, and Zhe Fan. 2017. Reducing Uncertainty of Probabilistic Top-k Ranking via Pairwise Crowdsourcing. *IEEE Transactions on Knowledge and Data Engineering* 29, 10 (2017), 2290–2303.

[27] Sean Monahan, John Lehmann, Timothy Nyberg, Jesse Plymale, and Arnold Jung. 2011. Cross-Lingual Cross-Document Coreference with Entity Linking.. In *TAC*.

[28] Christopher Re, Nilesh Dalvi, and Dan Suciu. 2007. Efficient top-k query evaluation on probabilistic data. In *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 886–895.

[29] Mehdi Sadri, Sharad Mehrotra, and Yaming Yu. 2016. Online adaptive topic focused tweet acquisition. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 2353–2358.

[30] Wei Shen, Jianyong Wang, and Jiawei Han. 2014. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering* 27, 2 (2014), 443–460.

[31] Wei Shen, Jianyong Wang, Ping Luo, and Min Wang. 2012. Linden: linking named entities with knowledge base via semantic knowledge. In *Proceedings of the 21st international conference on World Wide Web*. 449–458.

[32] Wei Shen, Jianyong Wang, Ping Luo, and Min Wang. 2013. Linking named entities in tweets with knowledge base via user interest modeling. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 68–76.

[33] Marina Sokolova and Guy Lapalme. 2009. A systematic analysis of performance measures for classification tasks. *Information processing & management* 45, 4 (2009), 427–437.

[34] Mohamed A Soliman, Ihab F Ilyas, and Kevin Chen-Chuan Chang. 2007. Top-k query processing in uncertain databases. In *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 896–905.

[35] Mohamed A Soliman, Ihab F Ilyas, and Kevin Chen-Chuan Chang. 2008. Probabilistic top-k and ranking-aggregate queries. *ACM Transactions on Database Systems (TODS)* 33, 3 (2008), 1–54.

[36] Alaa Tharwat. 2020. Classification assessment methods. *Applied Computing and Informatics* (2020).

[37] Vasilis Verroios and Hector Garcia-Molina. 2019. Top-k entity resolution with adaptive locality-sensitive hashing. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1718–1721.

[38] Wei Zhang, Chew Lim Tan, Yan Chuan Sim, and Jian Su. 2010. NUS-I2R: Learning a Combined System for Entity Linking.. In *TAC*.

[39] Xi Zhang and Jan Chomicki. 2009. Semantics and evaluation of top-k queries in probabilistic databases. *Distributed and parallel databases* 26, 1 (2009), 67–126.