Preference-based Twitter Analytics

Lena Rudenko and **Christian Haas** [1]

Abstract. Within the last decade Twitter has become one of the most important sources for information distribution for many people. At the same time, finding useful and interesting tweets on a specific topic remains a non-trivial task, because there are thousands of new posts every minute. In this paper, we describe our preference-based search approach on Twitter messages, which allows users to formulate their wishes in a flexible but intuitive way to get the best possible results. For this, we introduce a new `CONTAINS` preference constructor to search on full-text data, use NLP techniques to handle natural language mistakes, and present preliminary experiments.

1 INTRODUCTION

Social media have become an integral part of our live. Over the morning coffee, on the way to work, during lunch break and between exercises in the gym, we are increasingly viewing new photos on Instagram, posts on Facebook and the latest news on Twitter.

Today, social networks and particularly Twitter spread information faster than usual traditional media. The paradigms are changing: first there was the telegraph, then radio and television. Today it is the turn of internet and social networks. They are the fastest way to spread information. For example, the first message about the emergency landing of the flight AWE 1549 in the Hudson River appeared on Twitter less than 5 minutes after the actual event (see Figure 1). On Twitter you get the information fast and firsthand, before journalists process it and submit it in the convenient way, placing the "right" accents. Why watch CNN when you can follow Donald Trump's Twitter?

Figure 1. The first tweet about "Miracle on the Hudson".

However, it's not Mr. Trump alone, who makes Twitter alive. The emergency landing in Hudson River was reported by the ordinary users with a dozen followers from relatives and colleagues. Every second, about 8.8 thousand messages[2] are posted by Twitter users. Thus, having a source of almost unlimited information, we face the task of finding relevant and interesting content.

Twitter users usually follow some other users who post information, the followers are interested in, e.g., politicians, actors, journalists, sportsmen, someone from the professional field and so on. Tweets posted by the users you follow, show up in your feed automatically. But we do not know other millions of accounts that might post something important or relevant for us.

Twitter allows to browse all posted tweets using *keywords* or *hashtags*. Even quite complex expressions are possible, e.g., if you want to have all tweets (but no retweets) with the term *puppy*, you should enter in the search box the following query: *puppy -filter:retweets*. This is a great support for users looking for tweets outside the followed accounts, but with one big drawback - if no perfect match results are found, the user gets nothing back. An alternative to *hard constraints* are the *soft constraints* approach based on user *preferences*. Preference based queries allow the user to specify another option(s) in case the first choice has no results.

Obviously, preferences should be applied to *tweets*, which are essentially texts, although with many features. The challenge in connection with preference-based evaluation of tweets has two aspects:

- Literally anything can appear in the text. A tweet is limited only by the number of characters.
- Tweets are very special texts with typing errors, abbreviations, various spelling forms and other special language constructs.

Tweet analysis is a popular topic among scientists. One of the research directions, for example, is the analysis of user sentiment and publics' feelings towards certain brand, business, event, etc., cp. [21, 23, 18]. A content analysis of tweets on various topics is also often the focus of researchers, cp. [3, 4, 24]. But none of them deal with Twitter analysis using user preferences and NLP techniques to handle natural language mistakes.

Our goal is to develop an approach that makes it possible to search for interesting and relevant information among *all* posted tweets using preference based queries spezified by a user with the guarantee to get no empty result set if perfect hits do not exist. To achieve this goal and taking into account the aspects mentioned earlier, we plan to proceed as follows: instead of asking for a perfect match, we search for certain user-defined terms in the tweet text. With the expression $text$ `CONTAINS`$(('figureskating'), ('isu'))$, for example, the user is looking for all tweets including the term *figure skating*. If no such tweets exist, the messages including *isu* should be returned. And at this point we come across the second aspect: looking for "figure skating", we want to find also tweets with "figure sktng" (abbreviation) or "fogure skating" (typing error), too. In this aspect some methods from the field of Natural Language Processing could be helpful.

The remainder of this paper is organized as follows: Section 2 recapitulates essential concepts of our *preference model* and *tweet* as a text message. In Section 3 we introduce the developed `CONTAINS` *preference*, while in Section 4 we give a short overview of an *implementation* of this preference. Our results on *perliminary experiments* are shown in Section 5. Section 6 contains a *summary and outlook*.

[1] University of Augsburg, Germany, email: lena.rudenko@informatik.uni-augsburg.de and christian.michael.haas@student.uni-augsburg.de

[2] Tweets per second: internetlivestats.com/one-second/

2 BACKGROUND

In this chapter we give a brief overview on **preferences** and **tweets**.

2.1 Preference Model

Preference modeling have been in focus for some time, leading to diverse approaches, cp. [2, 10, 8, 1, 9, 5]. These works have in common the treatment of the differentiated user wishes, which prefer some results to the others. We use the approach of [11] that models **preferences** as strict partial order with intuitive interpretation *"I like A more than B"*. A preference is defined as $P = (A, <_P)$ on the domain of A. Thus $<_P$ is irreflexive and transitive. Some values are considered to be better than some others and the term $x <_P y$ can be interpreted as *"y is preferred over x"*. If two values cannot be ordered by the strict partial order $<_P$, they are called *indifferent*, i.e. $x \sim_P y \Leftrightarrow \neg(x <_P y) \wedge \neg(y <_p x)$.

The *maximal objects* of a preference $P = (A, <_P)$ on an input dataset R are all tuples that are not dominated by any other tuple w.r.t. the preference. These objects are computed by the preference selection operator $\sigma[P](R)$. It finds all best matching tuples t w.r.t. the preference P:

$$\sigma[P](R) := \{t \in R | \neg \exists t' \in R : t <_P t'\}$$

The evaluation follows a Best-Matches-Only (BMO) query model that retrieves exact matches if such objects exist and best alternatives else.

2.2 Preference Constructors

To express simple preferences targeting one attribute, diverse base preference constructors are defined. There are base preference constructors for numerical, categorical, temporal, and spatial domains, cp. [11]. In this work we focus on the categorical preferences. We want to perform a preference based search on *tweets* that exist as text and thus belong to the categorical attributes. But first we briefly describe the existing preference constructors on the categorical domains.

There exist the following categorical preferences: `LAYERED`, `POS/POS`, `POS/NEG`, `POS`, `NEG`. All categorical base preferences are sub-constructors of $\texttt{LAYERED}_m$ (cp. Figure 2). It divides the domain into $m + 1$ disjunct sets that form an ordered list. Each set consists of i values, where $i >= 1$ and can be different in different sets. The values in one set are considered as *substitutable*. The different sets have different satisfaction levels, whereby the values in the set with $m = 0$ are the *perfect* ones. To be sure that all domain values are covered by the preference constructor, one of the sets may be named *others*.

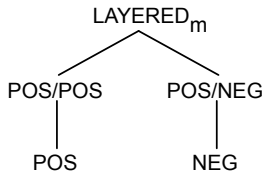

Figure 2. Taxonomy of categorical base preference constructors

The other categorical preferences can be derived from $\texttt{LAYERED}_m$, e.g., `POS/POS` is equal to $\texttt{LAYERED}_{m=2}$`(A, POS1-set, POS2-set, others)`, `NEG` is equivalent to $\texttt{LAYERED}_{m=1}$`(A, others, NEG-set)` and so on.

The existing constructors for the categorical attributes work quite well if the domains values are finite. For example, book titles, country names, animal species or car colors can be searched and evaluated with the categorical preferences. A preference query $\texttt{LAYERED}_{m=1}$`(color, {'white'}, others)` will return all objects that have a value *white* in the attribute *color*. If no such objects exist, objects of any color are returned. This approach will obviously not work with free text attributes, such as *comment*, *note* or *tweet*.

2.3 Tweets

A **tweet** in general is a complex object with many other attributes besides the actual *text*, such as *language* of the user profile, *number of responses* received by the message and so on. A complete list of all tweet attributes can be found on the Twitter developer website[3]. But when you talk about a tweet on a daily basis, you are referring to **text messages** posted by users from their personal Twitter accounts. In this paper, when we use the term *tweet*, we mean a text message only. The numerous attributes that each tweet object includes are not used in this work.

Tweets are rather short messages with the maximum length of 280 characters, which can contain not only words but also *links*, *special characters* (e.g. emojis), *references* to other user accounts and *hashtags* (keywords or phrases, which help users to find relevant topics).

Figure 3. A simple tweet example

In Figure 3 you can see a simple example of a tweet posted by the user *c_haasem*. It includes the hashtag *#science*, link to the website of a chair at the University of Augsburg and a short sentence with slang abbreviation *Its*, typing errors *preferncе* and two emojis.

All this is very typical for tweets, which is not surprising when you consider that 83%[4] of all users are those who use the mobile Twitter application and post fast without distracting from other activities.

Not only the content diversity of the tweets, but also various forms of their terms (*its*, *it is* or *it's* as variants of the same expression) do not allow the existing categorical preferences to provide acceptable results on these text messages. However, based on the existing preference constructors, we decided to develop an additional one that will be able to find the tweets taking user preferences into account.

3 CONTAINS PREFERENCE

This section is about our new `CONTAINS` preference. We start with a formal description of the `CONTAINS` *constructor* and explain how the *score value* for this preference is calculated. Then we describe the steps and associated techniques from the field of Natural Language Processing we apply to the tweets during the *preprocessing phase*. In conclusion, we discuss in which case we can consider two terms as *similar*.

[3] Twitter Developer Website: www.developer.twitter.com

[4] Statista: de.statista.com/statistik/daten/studie/541918/umfrage/anteil-der-mobilen-monatlich-aktive-nutzer-von-twitter-weltweit/

3.1 CONTAINS Constructor

Our new preference is called `CONTAINS` and has a similar structure as `LAYERED`. For `CONTAINS` preference a user defines some levels, each with a set of words or terms he would like to see in the result set. The general idea is as follows: If the text message **contains** some term from one of the levels, it gets a number value corresponding to the level number the term belongs to. The smaller the value, the more preferred the tweet is. Finally, the tweets with the smallest values belong to the evaluation result set.

Let $L = (L_1, ..., L_m)$ with $m > 0$ be an ordered list of m sets with terms t_i defined by a user as those he wants to see in the result set. Additionally there is also another level with the set called *others*. All terms that will be found in tweets but are not listed in the sets of levels $1, ..., m$ belong to the $others$-set. All sets are disjoint, each term t_i belongs to one set only. The terms within a set are indifferent. Assume x and y are two terms from a set of all terms that occur in tweets. Thus the following applies:

$$x_i <_P y_j,\ x_i \in L_i,\ y_j \in L_j,\ j < i\ for\ i, j \in \{1, ...m + 1\} \quad (1)$$

$$x_i \notin L_1, ..., L_m \Rightarrow x_i \in others \quad (2)$$

$$x \wedge y \in L_i \Leftrightarrow (x \sim_P y) \quad (3)$$

For each considered tweet some *score value* is calculated. The *lower* the score value for a message, the *more preferred* it is. A tweet can contain terms belonging to *different levels*.

Definition 1 Score Function
Let $t_1, ..., t_n$, $(n > 0)$ be terms in some tweet t that belong to some different levels $L_{1,...,m+1}$. The score function is defined as:

$$f(t) = \min(\{i - 1 | i = 1, ..., m + 1\})$$

Example 1 The score value for the following tweet t := *"After @FSUfigure started their campaign against ISU's unfair judging, @olympicchannel removed TES scoreboard from the figure skating live broadcast at the Youth Olympics"* should be calculated. Regarding the preference `P:= t CONTAINS` $(('figureskating'), ('isu'))$, t has the score value $f(t) = 0$, even though the text includes terms from levels L_1, L_2 and from *others* (here L_3).

3.2 Natural Language Processing

As mentioned in Section 2.3, tweets are short text messages with a high percentage of typing errors, abbreviations and other language features. We have incorporated some techniques from the field of Natural Language Processing into a preprocessing step to map different spelled or misstyped terms to the terms defined by the `CONTAINS` preference.

Natural Language Processing (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence that involves a wide collection of techniques for language analysis and language development with the aim to achieve the most natural result as possible [13].

The collection of NLP techniques can be divided into *syntactic* and *semantic* methods, depending from information level they handle (cp. [6]). *Semantic* methods refer to the *meaning* of input (cp. [14]): Answering user questions automatically, summarizing the topic of a document, Named Entity Recognition (finding personal names and locations in texts) are some examples [13, 6]. *Syntactic* methods, on the other hand, deal with the *structure* of a word, sentence or even complex text [14]. They include, for example, lemmatization, stemming, parsing, part-of-speech (POS) tagging and the segmentation of a text into sentences or words [6, 16]. Parsing describes the segmentation of words in a sentence into subject, predicate and object. POS tagging assigns each word to its part of speech, while lemmatization and stemming reduce a word to a basic form.

The new `CONTAINS` preference should be able to find the tweets where the searched terms are written incorrectly or different. For this we will use some of the previously mentioned *syntactic methods* from NLP. To keep the effort of the work within reasonable limits, we have concentrated on the tweets in English.

For now, we have concentrated only on *nouns*, which are naming words and identify persons, places, things, animals, feeling, ideas, things, events, substances, or qualities. The proposed approach can be extended with little effort to a few more parts of speech, such as verbs, adjectives, adverbs etc. Note that the nouns are also expected as input terms for `CONTAINS` preference.

The following steps are applied to a text before a score value can be calculated with regard to the preference: The tweet text is *normalized*, *segmented* into *sentences*, which are further divided into *single words*. For these words, a *part of speech* is determined and the *nouns* are sorted out of the whole set. The nouns will be stemmed in the last step. A graphic sequence of this process as a pipeline can be seen in Figure 4.

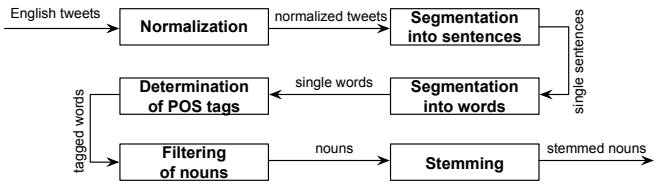

Figure 4. Pipeline of tweets preprocessing steps.

Let us take a closer look at the individual steps:

1. **Normalization.** In this step, *emojis* and *links* are removed from the tweet text, as they are an obstacle to part-of-speech definition and obviously cannot be specified as terms in the preference. If the tweet is a retweet, the *retweet-shortcut* (RT at the beginning of the message) and the *@username links* are deleted. The *@username mentions* at the beginning of the tweet (if the tweet is a reply to one or more users) are also removed. In addition, each hashtag term is cleared of the hash character #. Figure 5 shows a tweet before and after normalization.

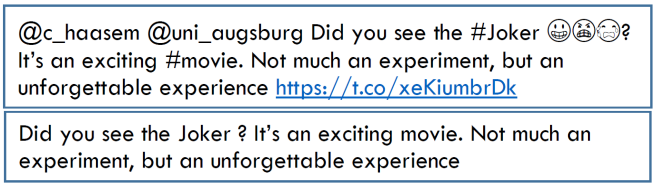

Figure 5. Tweet before and after normalization.

2. **Segmentation into sentences.** After normalization, the text is divided into individual sentences. Although tweets usually contain very few sentences, this segmentation is still important because

the POS Tagger determines the word types of the individual words based on the sentence structure.

3. **Segmentation into words.** In this step the sentences are segmented into individual words and passed on to part-of-speech tagger.
4. **Determination of POS tags.** Each input word is assigned to its part of speech with a part-of-speech tagger. Most such POS taggers use statistical methods to determine a part of speech. They are trained on the data where each word has already been tagged. The trained tagger then assigns a new input word the tag that has the highest probability in the context (cp. [14]). In Table 1, a sentence *"Gustav likes sunny days to go swimming"* is tagged with a Penn Treebank Tagset [15]. This tagger for English language is also used in our implementation.

Gustav	likes	sunny	days	to	go	swimming
NNP	VBZ	JJ	NNS	TO	VB	VBG

Table 1. A tagged example sentence.

The tags determined by the tagger can be understood as follows: *NNP* is a proper noun in singular, *VBZ* is a verb in the present tense and in the third person singular, *JJ* stands for an adjective, *NNS* for a noun in the plural, *TO* is a word "to", *VB* corresponds to a verb in infinitive and *VBG* to a verb in the gerund.

5. **Filtering of nouns.** Penn Treebank Tagger distinguishes general nouns and proper nouns in singular and plural. We need all four groups, because, for example, both *"Trump"* and *"president"* can be used in the `CONTAINS` preference. In this step the filtered nouns are additionally cleared of unnecessary characters. *Points* and *apostrophes* remaining in words are removed, as they make stemming problematic and complicate comparision with the terms from the preference during evaluation.
6. **Stemming.** After cleaning the words from special characters, at first it is ensured that the word is in lower case and then a stemming is performed. Stemming shortens a word back to a word stem common to all morphed forms [22]. Stemming should not be confused with *lemmatization*. These two processes reduce words to a certain basic form, but while lemmatization returns the word to its dictionary form, stemming often creates an unnatural common form. Lemmatizers are often used when the produced lemma has to be used in the area of word meaning or generally a natural real basic form is needed. We use stemming precisely because it takes no account of the meaning of the word. Since stemming create the unnatural common form anyway, it is possible to stem the misspelled words too.

   The most common stemmer for the English language, which we also use in our work, was developed by M. Porter and published in [20] in 1980. A schematic representation of the form $[C](VC)^m[V],\ m \geq 0$, where C is a consonant, V - a vowel, is calculated for each word. Then the word stem is calculated by changing or eliminating the current word ending in several steps according to the appropriate rules.

3.3 Edit Distance

The stemmed nouns from the tweets should now be compared with the nouns specified by the user in the `CONTAINS` preference. Note that the preference nouns are also stemmed by the same procedure to enable a fair comparison. Ideally, the word (stem) from a tweet is exactly the same as the word (stem) from a preference. But we know that tweets are rich with spelling mistakes. In order to allow comparison of the misspelled nouns, we calculate the distance between a stemmed input noun and a stemmed noun from the tweet. The terms with the distance below a certain value are regarded as equal.

For the comparison of words that include spelling mistakes there is a wide range of techniques, which are based on syntactic and statistical methods (cp. [12]). In this work we use the **Damerau-Levenshtein** algorithm that works over an *edit distance*. Edit distance between two words is defined as number of changes in one word that have to be made to get from that word to another one. Changes include *inserting* and *deleting* of a character, *replacing* of a character with another one, and *swapping* the positions of two characters that cover the most common spelling mistakes.

Damerau [7] and Peterson [19] independently found out that about 80% of all spelling mistakes belong to one of the four groups shown in Table 2.

error	**incorrectly**	**correctly**
swap of two adjacent characters	huose	house
one letter false	dok	dog
one letter missing	aple	apple
one letter to many	informaition	information

Table 2. The four most common spelling mistakes [7].

The extended variant of Damerau-Levenshtein algorithm has a *threshold value*. The calculation is aborted if the distance between two words is too big and the threshold value is exceeded. Aborting the distance calculation is possible because we are not interested in the distance between two words, but in very similar words. This results in runtime advantages.

We have to define the term *similar words*. We need to determine if, for example, *"poll"* and *"pole"* - two different correctly spelled words - are similar. We have also to regulate if *"skater"* and *"skating"* - two different nouns with the same stem - are similar words in our approach. Finally, we need to determine whether misspelled words are similar with the correctly spelled ones, e.g., *"schedule"* and *"shedule"*. In addition, we should consider whether the degree of similarity is related to the length of the word. After all, a long word offers more room for possible errors.

We have decided to consider three possible scenarios:

1. The input word t from the `CONTAINS` preference is **short**: $|t| \leq 4$. This length was taken for several reasons: In [12] the author defines short words as word with a maximum length of 4 characters. Moreover, the distance is determined not for the original words, but for their stemmed variants, which in many cases are shorter. The allowed distance d between the short input word and the word from tweets to consider them as equal has to be $d = 0$, so they must be *identical*. If the distance is greater, you will identify the pairs *"cat"* and *"cap"* or *"poll"* and *"pole"* as identical, which is obviously a mistake.
2. For words of **medium** length the distance of $d = 1$ is allowed. Medium-length words include words with more than 4 but less than 8 characters: $4 < |t| < 8$. This decision is supported by the work of Norvig [17]. He analyzed the number and length of the words that appear in books scanned by Google Books. He found that 80% of written words are between 2 and 7 characters long and the average length is about 4.7 characters per word.

   Distance 1 means that *one change* is enough to make the input word and tweet word exactly equal. The possible changes correspond to the 4 most common spelling mistakes listed in Table 2. In

the words of medium length (note that the length is always determined for a stemmed word) the probability of making a mistake is much higher than in the short words. At the same time, it is rather improbable to form an existing correct word by making only one change: inserting, deleting, replacing of a character or swapping the two characters positions.
The described approach will define equality for a pair of words *"schedule"* and *"shedule"*. The *correctly* spelled word *schedule* will be reduced by he stemmer to the form *schedul* with the length of 7 characters. The *incorrectly* spelled word *shedule* will be stemmed to *shedul*. The edit distance between these stems is $d = 1$ (inserting of *c* into *shedul*), hence the words are *"equal"* and the tweets containing *"shedule"* will also belong to the result set, if the preferred term in `CONTAINS` is *"schedule"*.

3. The last scenario applies to **long** words with at least 8 ($|t| \geq 8$) characters after stemming. This group is relatively small: According to Norvig (cp. [17]), only 20% of all words reach the length of 8 characters or more. But he examined the words in the form they appeared in the books. Stemming usually shortens the words, so the percentage for this group is clearly below 20%. This is also supported by the fact that tweets are limited in length to 280 characters and users try to express their thoughts briefly and clearly. The short and precise words are a part of it.
For the small part of the long words we allow the distance of $d = 2$. It is not difficult to make a mistake in a long (and often complicated) word. Especially because tweets are usually written casually, parallel to another activity. Because of the generally low number of long words, the danger of finding two of them within the distance of $d = 2$ is also low.

To summarize this section: The `CONTAINS` preference allows the user to list the terms that he would like to see in the tweets. A certain deviation in spelling or form, as well as possible errors should be taken into account as much as possible. The texts from the tweets are preprocessed using some methods from Natural Language Processing. Then, both the input terms and the terms from the tweet are stemmed. Finally, the build stems are compared, and if they are equal (which is defined differently for terms of different lengths), the tweet belongs to the result set.

4 IMPLEMENTATION

The implementation of the `CONTAINS` preference was done in Apache Flink. *Apache Flink*[5] is an open source framework for the analysis and processing of data streams in real-time. Apache Flink has interfaces for Java, Skala, Python and SQL. Among others, the advantages of Apache Flink are the processing of large amounts of data in real-time, the generally high performance and the high fault tolerance.

The preference core are the so-called *levels* - the sets of terms that users want to see in a tweet. The *lower* the level of the set is, the *more preferred* are the terms in this set. The levels have numbers, the numbering starts with 0 for the most preferred level. The number of levels and the number of terms in the levels is not limited but each additional level and term also means an increase in runtime.

Levels are stored in a map structure: *key* is an object that contains all elements of a level. The *value* indicates the number of the level. The levels are defined by the user and assigned to the `CONTAINS`-preference in the program call. The order of insertion is important: The earlier a level is inserted, the more preferred it is. To keep the order of levels insertion, we use a *LinkedHashmap* structure. If we are sure that the first element in map structure is the best one, the second - the second best one and so on, the calculation of the score value for each tweet can be stopped as soon as there is a match between an element of the current level and the term from the current tweet. The score value is then the level number of the current set.

The number of comparisons between preference terms and nouns in the tweet is then between 1 in the *best case* (the first element of the most preferred set corresponds to the first noun of the tweet) and $l * p$ in the *worst case*, where l is the number of nouns in the tweet and p is the number of LinkedHashmap entries. The worst case occurs if no noun from any of the levels of the `CONTAINS`-preference occurs in the tweet.

When checking whether the `CONTAINS` preference term is contained in a tweet, we can set the variable *levenshtein* to *true* or *false*. This variable determines whether the possible spelling mistakes in the tweets nouns should be considered (cp. Section 3.3). This allows us, among other things, to compare the results both from the quality and from the runtime point of view.

The calculation of the *score value* of a tweet takes place in the method *getScore* and depends on the value of the variable *levenshtein*. If it is *true*, the length of the stemmed preference term is considered. Is the stemmed term t long enough, 1 (for $4 < |t| < 8$) or 2 (for $|t| \geq 8$) possible spelling mistakes are allowed. In this case we use a *Damerau-Levenshtein* algorithm to compare the preference term and the nouns of the tweet. Using the algorithm version with a threshold allows us to interrupt the calculation if the allowed distance (1 or 2) is exceeded. This improves runtime.

If the stemmed term t is too short ($|t| \leq 4$) or we do not want to accept misspelled words (*levenshtein* is *false*), the standard function *equals* will be used for comparision. The stemmed preference term and the stemmed noun from the tweet must match completely, only then the current tweet belongs to the result set.

For the necessary NLP steps we use an *Apache OpenNLP*[6] library.

5 EXPERIMENTS

In this section we describe preliminary test results. These tests should give us an impression of how *efficient* the developed approach is (runtime) and what *quality* the delivered results have (do the results correspond to the query and whether something remains undiscovered).

For comparison we have implemented another method (further called *methodB*) of score calculation for `CONTAINS` preference. For this, there is no complex natural language tweet preprocessing. We only remove the initial signature of retweets *"RT @name"*, the rest of the text remains unchanged. We use a pre-implemented case insensitive function *contains* from the *Apache Commons Lang 3.9* library that checks if one term is contained in another one. For each term in each of preference set it is checked whether it occurs in the tweet and if this is the case, the number of the term's level is assigned to the tweet. The total tweet's score is then a *minimum* of all assigned level numbers (cp. Definition 1).

Although we do not preprocess tweets using *methodB*, stemming for the preference terms is done anyway. Thus *methodA* (our approach with complex preprocessing and spelling correction) compares stemmed preference terms with the stemmed tweet's nouns chosen while preprocessing of the text. *methodB*, which we use as a reference point, compares the stemmed preference terms to all original written words (nouns or no) from the tweet's text. The reason for

[5] Apache Flink: flink.apache.org

[6] Apache openNLP: opennlp.apache.org

this decision is due to the way the plural of some nouns are formed in English. The word ending *-y* in singular changes into *-ies* in plural, e.g., *study* and *studies*. As a consequence, when searching for the word *study* in the tweet text, *studies* will not be considered as a hit and vice versa. Spelling mistakes are not taken into account in this method and only the correctly spelled words are regarded as hits.

5.1 Quality Tests

To test the quality of the returned results, 10.000 completely random tweet objects were saved to a text file. As input terms, we used both words that changed their form after stemming and those that remained unchanged. In addition, we selected words of different lengths to be able to test whether wrong spelled words will be found (cp. Section 3.3).

The two methods are applied to the tweets that we have stored. Table 3 shows the *number* of hits (tweets containing the corresponding term). In Table 4, we see the same results expressed in terms of ***Precision (P)*** - the percentage of retrieved documents that are relevant to the query - and ***Recall (R)*** - the percentage of the relevant documents that are successfully retrieved.

term	**stemmed term**	**methodA**	**methodB**
sandwich	sandwich	1	2
housing	hous	35	93
skating	skate	25	1
decision	decis	7	7
connection	connect	8	20
replacement	replac	10	15
development	develop	13	19
independence	independ	6	8
forest	forest	6	7
bear	bear	6	20

Table 3. Number of hits using two different methods.

term	***P*** methodA	***P*** methodB	***R*** methodA	***R*** methodB
sandwich	100%	100%	50%	100%
housing	100%	39%	97%	100%
skating	4%	50%	100%	100%
decision	100%	100%	100%	100%
connection	87.5%	40%	78%	89%
replacement	40%	27%	100%	100%
development	100%	74%	93%	100%
independence	50%	37.5%	100%	100%
forest	100%	100%	85.7%	100%
bear	100%	30%	100%	100%

Table 4. Precision (P) and Recall (R) for two different methods.

We see that the number of hits in Table 3 for the *methodB* is for most terms higher than that of the *methodA*. There is only one exception for the term *skating*, which we will explain later. *methodB* returns on average more tweets. Moreover, *recall* of *methodB* is almost everywhere 100%, i.e. *all relevant tweets* have been found. The only exception we observe for the term *connection*. One of the relevant tweets includes this misspelled term (*konnection*) and will not be found by *methodB*, but will be present in the results of *methodA*. *methodA* shows slightly worse recall results, but the difference doesn't exceed 15%, except for one term - *sandwich*. *methodB* returns twice as many relevant tweets. But if you look at Table 3 with the number of hits, you can see that twice as many means 2 tweets, instead of 1, which *methodA* finds.

The reason for this is that our approach takes into account only a very small number of *hashtags*. They can be almost any - complex, non-existent, invented: *#Beginner'sMind*, *#BuildingConnections*, *#ManagersDay2016* are only some examples. Obviously, that stem *sandwich* is found in the hashtag *#tunasandwich* by *methodB* but not by *methodA*, which can not split this complex term into its components. This is also the case for stem *forest* (cp. Table 3): the seventh hit of *methodB* is a tweet with the hashtag *#UgandasForests*, which remains undetected by *methodA*.

Besides the "good" results, *methodB* also detects the "bad" ones in this way: the tweets with hashtags *#clubhousegh* and *#DaquansPlayHouse* are e.g. returned when the user is looking for *housing*. Let's take a closer look at this term. Since the length of the stemmed term is $|hous| = 4$, no different spelling or mistakes are allowed. The tweets delivered by *methodA* (our approach) contain the words: *house*, *housing*, *House*, *house's*, which all make sense. In the tweets delivered with *methodB*, we find other terms besides those already mentioned, e.g. ***hous****wife*, *pent****house***, ***thous****and*. Also the proper names (Whitney ***Hous****ton*, Amy *Wine****house***), hashtags (#*uk****hous****ing*, #*club****house****gh*, #*DaquansPlay****House***) and account names (@*RachaelL****Hous***, @*Alert****Hous****ton*, @*6Vis****house*** etc.) are included. The result is understandable, because all listed words contain stem *hous*. But it is not what we wanted.

Another reason for the larger result of *methodB* are *verbs*, *adjectives* or even other *nouns* with the corresponding stem (e.g. to ***connect*** and ***connect****ed* while searching for ***connect****ion*; ***bear****y*, to ***bear*** or ***bear****d* - for ***bear***).

We should not forget words whose meaning changes to the opposite just by adding a prefix too: *dependence* and *independence*, *spelling* and *misspelling*. While looking for the first term of each pair, *methodB* will finde also the second one. And since their meanings are completely opposite, the presence of the latter terms (with prefix) in the result set is undesirable if the user wants the former ones.

The returned non-relevant results have a great effect on the *precision* values, which we can see in Table 4 for the *methodB* very well. Precision values of *methodB* are almost always worse, sometimes very clearly than those of *methodA*. The only exception is the term *skating*. This is the only one term for which the number of hits of *methodA* is higher and precision is lower than for *methodB* (see Tables 3 and 4). If we look more closely at the resulted tweets, we do not see in the texts *skating*, *skate*, *skater*, etc. that were expected. Instead we find *state* and/or *Kate*. The reason is the allowed deviation of the correct spelling / error correction. The original term *skating* is stemmed to *skate*. It has a length of 5, so there is a distance of 1 between *skate* and the stemmed nouns from the tweet allowed (cp. Section 3.3). *Skate* and *state* have one different letter, *skate* has one letter more than *Kate* (the approach is case insensitive), so the result is quite logical.

Our approach has also found a tweet with a misspelled term *konnection* (instead of *connection*). Unfortunately, the number of incorrectly returned tweets is too high with this correction. So we need to revise this step of our approach. We can increase the length of the words that are allowed to be corrected. The implementation would be very simple, but we cannot guarantee that even with the longer words, a very similar but completely different term will not appear at the distance of one change.

Another option is to create some kind of lexicon where the most common (spelling) variations of the words are collected. A big advantage of this approach is that we do not have to limit ourself to 4 most common spelling mistakes (see Table 2), but can also include

the very popular abbreviations (e.g. *prof* for *professor*). A big disadvantage in return is the effort required to create this lexicon.

In general, we can say that the quality of results delivered with *methodA* is higher compared to *methodB*. But spelling mistake correction / different spelling part must clearly be rethought and reimplemented.

5.2 Runtime Tests

We also did some runtime tests: we compared the time *methodA* and *methodB* take to evaluate *one* tweet. The tests were performed on Intel® Xeon® Scalable Processor "Skylake" Silver 4110 with 2.10 GHz, 192GB DDR4 and 2x 4TB SATA3-HDD. An installed softwaresystem is Ubuntu Linux 18.04 LTS 64 bit.

For the tests we collected real tweets. Each file contains a different number of tweets (cp. Table 5). The tweets were collected live and are different in each file. All tweets in each file were evaluated with respect to two preferences: `P1:= text CONTAINS`$(('bear'), ('forest'))$ and `P2:= text CONTAINS`$(('housing'), ('decision','statement'), ('connection'))$ using *methodA* and *methodB*. The results can be found in Table 5 for P_1 and Table 6 for P_2. The measured time includes both the tweets *preprocessing* and the actual *score calculation*. As you can see, the runtime of *methodB* is always significantly

file	tweets number	time methodA	time methodB
f1	120 277	374.09	108.07
f2	133 278	446.79	116.8
f3	268 231	438.76	115.39
f4	317 816	443.53	116.13
f5	399 929	369.35	106.42
f6	599 275	365.1	105.3
f7	931 027	431.88	113.99
f8	1 160 599	435.99	114.41
f9	1 307 764	362.31	104.46
f10	3 592 899	348.25	104.3

Table 5. Runtime of two different methods per tweet in μs for P_1.

file	time methodA	time methodB
f1	374.4	110.65
f2	449.62	119.62
f3	441.18	118.32
f4	447.05	118.79
f5	370.8	109.17
f6	365.71	107.85
f7	434.53	116.06
f8	438.24	117.31
f9	362.58	106.48
f10	349.26	105.66

Table 6. Runtime of two different methods per tweet in μs for P_2.

better compared to *methodA*. But let us not forget that our approach (*methodA*) requires much more complex preprocessing and score calculation. The other thing that stands out in the results is that regardless of a preference, the runtime for both methods remains very stable and do not depend on the file size (number of tweets).

It seemed interesting to us that the results can be clearly divided into two groups: the files *f1*, *f5*, *f6*, *f9* and *f10* belong to *Group* 1 with a *lower* runtime per tweet, the files *f2*, *f3*, *f4*, *f7* and *f8* belong to *Group* 2 with the *higher* runtime. And this for both preferences and both methods. The only reasonable explanation is that the tweets in the first group are on average shorter. This means that fewer comparisons are required (no matter which method is used), which shortens the runtime.

In Table 7 we have summarized for both preferences P_1 and P_2 the runtime for complete tweet evaluation (preprocessing and score calculation) and the runtime for separate score calculation. Score calculation is about 60-65% of the total runtime. We will use this information to improve our approach in the future.

file	full P_1	getScore P_1	full P_2	getScore P_2
f1	374.09	235.3	374.4	235.3
f2	446.79	292.59	449.62	295.19
f3	438.76	288.5	441.18	292.2
f4	443.53	295.57	447.05	295.26
f5	369.35	233.22	370.8	235.59
f6	365.1	231.51	365.71	233.83
f7	431.88	284.79	434.53	286.8
f8	435.99	288.1	438.24	289.83
f9	362.31	229.64	362.58	230.72
f10	348.25	220.82	349.26	222.97

Table 7. Full and getScore runtime in μs for P_1 and P_2.

6 CONCLUSION

In this paper we have described our preference-based approach for tweets search. We have developed and implemented a new preference: `CONTAINS`. It allows users to specify the terms that are preferred in the result set, in layers: starting with the most preferred ones and descending further. We have also made some attempts towards the recognition of misspelled words, because spelling mistakes are very typical for tweets.

The first experiments have shown that our approach gives very good *precision* results in most cases. The misrecognized tweets are the result of the misspelling correction. This aspect clearly needs to be revised. *Recall* numbers are fine, but a few steps can be taken to improve them. For example, the *hashtags*, require special treatment, since in most cases they are not real words existing in the language and therefore cannot be treated like normal nouns.

Although we already had some thoughts on improving the runtime while implementing our approach, the experiments have shown that there is still a lot of work to be done in this direction.

In summary, we believe that the proposed approach has a lot of potential to analyze Twitter data, which we will extend and improve extensively in the future.

REFERENCES

[1] R. Agrawal and E. L. Wimmers, 'A Framework for Expressing and Combining Preferences', *SIGMOD Rec.*, **29**(2), 297–306, (may 2000).

[2] K. J. Arrow, 'Rational Choice Functions and Orderings', *Economica*, **26**(2102), 121–127, (1959).

[3] J. W. Ayers, E. C. Leas, J. Allem, A. Benton, M. Dredze, B. M. Althouse, T. B. Cruz, and J. B. Unger, 'Why do People Use Electronic Nicotine Delivery Systems (Electronic Cigarettes)? A Content Analysis of Twitter, 2012-2015', *PLOS ONE*, **12**(3), 1–8, (march 2017).

[4] P. Cavazos-Rehg, M. Krauss, S. Costello, S. Connolly, C. Rosas, M. Bharadwaj, and L. Bierut, 'A Content Analysis of Depression-related Tweets', *Computers in Human Behavior*, **54**, 351–357, (jan 2016).

[5] J. Chomicki, 'Querying with Intrinsic Preferences', in *Proceedings of the 8th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '02, pp. 34–51, Berlin, Heidelberg, (2002). Springer-Verlag.

[6] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. P. Kuksa, 'Natural Language Processing (almost) from Scratch', *CoRR*, (2011).

[7] F. J. Damerau, 'A Technique for Computer Detection and Correction of Spelling Errors', *Commun. ACM*, **7**(3), 171–176, (mar 1964).

[8] T. Gaasterland and J. Lobo, 'Qualified Answers That Reflect User Needs and Preferences', in *VLDB*, Santiago de Chile, Chile, (1994).

[9] V. Hristidis, N. Koudas, and Y. Papakonstantinou, 'PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries', *SIGMOD Rec.*, **30**(2), 259–270, (may 2001).

[10] R. L. Keeney and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Trade-Offs*, Cambridge University Press, 1993.

[11] W. Kießling, 'Foundations of Preferences in Database Systems', in *Proceedings of VLDB '02*, pp. 311–322, Hong Kong SAR, China, (2002). VLDB Endowment.

[12] K. Kukich, 'Techniques for Automatically Correcting Words in Text', *ACM Comput. Surv.*, **24**(4), 377–439, (dec 1992).

[13] E. D. Liddy, 'Natural Language Processing', (2001).

[14] S. Linckels and C. Meinel, *Natural Language Processing*, 61–79, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[15] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, 'Building a Large Annotated Corpus of English: The Penn Treebank', *Computational Linguistics*, **19**(2), 313–330, (1993).

[16] J. McMahon and F. J. Smith, 'A Review of Statistical Language Processing Techniques', *Artificial Intelligence Review*, **12**(5), 347–391, (oct 1998).

[17] P. Norvig. English Letter Frequency Counts: Mayzner Revisited or ETAOIN SRHLDCU, 2013.

[18] V. S. Pagolu, K. N. Reddy, G. Panda, and B. Majhi, 'Sentiment analysis of Twitter data for predicting stock market movements', in *International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, pp. 1345–1350, (oct 2016).

[19] J. L. Peterson, 'A Note on Undetected Typing Errors', *Commun. ACM*, **29**(7), 633–637, (jul 1986).

[20] M. F. Porter, 'An Algorithm for Suffix Stripping', *Program*, **40**, 211–218, (1980).

[21] H. Saif, Y. He, and H. Alani, 'Semantic Sentiment Analysis of Twitter', in *The Semantic Web - ISWC 2012*, eds., P. Cudré-Mauroux, J. Heflin, E. Sirin, T. Tudorache, J. Euzenat, M. Hauswirth, J. X. Parreira, J. Hendler, G. Schreiber, A. Bernstein, and E. Blomqvist, pp. 508–524, Berlin, Heidelberg, (2012). Springer Berlin Heidelberg.

[22] A. Samir and Z. Lahbib, 'Stemming and Lemmatization for Information Retrieval Systems in Amazigh Language', in *Big Data, Cloud and Applications - Third International Conference, BDCA 2018, Kenitra, Morocco, April 4-5, 2018, Revised Selected Papers*, pp. 222–233, (2018).

[23] V. Subramaniyaswamy, R. Logesh, M. Abejith, S. Umasankar, and A. Umamakeswari, 'Sentiment Analysis of Tweets for Estimating Criticality and Security of Events', *Journal of Organizational and End User Computing*, **29**, 51–71, (oct 2017).

[24] J. Sutton, S. Vos, M. Olson, C. Woods, E. Cohen, C. Gibson, N. Phillips, J. Studts, J. Eberth, and C. Butts, 'Lung Cancer Messages on Twitter: Content Analysis and Evaluation', *Journal of the American College of Radiology*, **15**, (nov 2017).