Twitter Hashtag Segmentation

Jack Reuter — Wesleyan University — jreuter@wesleyan.edu

***Abstract*—This paper describes an effort to segment non-delimited strings of English text, specifically hashtags. Extensive research has been done in word segmentation, particularly in languages like Chinese, for its lack of spacing between words, and German, for its extensive use of compounds. English, however, has been relatively untouched. The goal of this research is to adapt and extend methods used in prior research to fit the demands of modern English.**

I. Introduction

#wordsoftheday ... word soft he day?
#statefarmisthere ... state far mist here?
#brainstorm ... bra in storm?
#doubledown ... do u bled own?
#votedems ... voted ems?

WORD segmentation is an important first step in natural language processing (NLP). It is difficult to derive accurate meaning from a piece of text without first determining the words that it comprises. Twitter, for those unfamiliar, is a social media site that allows users to share brief (less than 140 character) "tweets" with their followers. In their tweets, users have the option of including hashtags, a form of metadata labeling. Typically, hashtags contain no delimiters between words. From these hashtags, Twitter compiles lists of the most noteworthy topics of the day, month, year.

Despite the abundance of garbage inherent in social media, trending hashtags and their related posts often follow relevant topics and provide insight into public opinion. Twitter mining has applications in public health [1], political sentiment [2], and emergency coordination [3]. The analysis of hashtags and the relationships between them is a bountiful area of research. The goal of this project is to further the reach of such research by enabling machines to process individual hashtags into understandable chunks of data.

II. Problem Definition

Formally, the problem is as follows: given a string $a_1a_2...a_k$, where each a_i is a meaningful substring (i.e. a word, name, abbreviation, etc.), determine where the boundaries lie between each a_i and a_{i+1}. For example, given a string such as "randompicoftheday" return "random pic of the day". Initially it may seem like a simple problem. Simply loop through all substrings of the input, looking for matches in a dictionary. Once all matches have been found, segment the string accordingly.

The real problem, however, lies in the phrase "then segment accordingly." From Xue 2003, "The key to accurate automatic word identification...lies in the successful resolution of these ambiguities and a proper way to handle out-of-vocabulary words" [5]. Although referring to the segmentation of Chinese characters, the sentiment is still very much appropriate. In our case, ambiguities occur when a string has more than one meaningful segmentation, out-of-vocabulary words when our dictionary fails us. Both scenarios occur frequently, and the success of our methods depends on the handling of such situations.

Consider, for example, the string "brainstorm". Using a table lookup, a machine could read this as either "bra in storm", "brain storm", or the correct, untouched "brainstorm". Adopting the jargon of Webster and Kit [4], we will refer to this as *conjunctive ambiguity*, i.e. when a meaningful string can be broken up into n meaningful substrings. The natural solution of course is to take the segmentation with the largest matched word. This maximum matching approach handles conjuctive ambiguity very well, for it is unlikely that a syntactically sound clause happens to merge into a larger word, yet it is quite common for a larger word to break up into dictionary-matchable pieces.

Maximum matching fails, however, in cases of *disjunctive ambiguity*—the situation when a string "ABC" can be broken up meaningfully as either "AB C" or "A BC". "doubledown", for example, could be interpreted as either "doubled own" or the correct "double down" (or even "do u bled own" which exhibits both conjunctive and disjunctive ambiguity). Taking the maximum matching here would result in the incorrect segmentation "doubled own". Disjunctive ambiguity, it appears, requires a bit more syntactic knowledge to resolve.

The other main issue is the handling of strings outside of out dictionary. The string "votedems", for instance, should be returned as "vote dems" and not "voted ems", while our dictionary may only contain the abbreviation "ems" and not "dems". With typos, abbreviations, online slang, and just a general abundance of linguistic rule-breaking in hashtags, these situations are bound to occur, and occur frequently. For this project to succeed, such unknowns must be recognized and handled appropriately.

III. Related Work

Since English text is almost always whitespace delimited, little to no work has been done in English word segmentation (excluding morphological segmentation). Such research, however, has been widespread in other languages where segmentation is an important first step for NLP. German and Chinese, for example, due to their large compounds and lack of delimitation, respectively, have been researched for years, yielding successful results. The following is a brief compilation prior methods which have been both succesful in their own rights and adaptable, in parts, for English usage.

1) *Parallel Corpora:* In [7], Koehn and Knight use data from monolingual and parallel corpora to learn splitting

rules for German compounds, achieving 99.1% accuracy. After first obtaining all possible segmentations of a compound into known words based on a German corpus, they choose most-likely segmentations by examining word frequency in that same corpus (favoring more common words), as well as word occurrence in a parallel English corpus, (favoring segmentations whose translations contain the same words as the translation of the original string). Part-of-speech (POS) tags are also taken into account to avoid splitting off suffixes and prefixes from root lemmas.

2) *Unknown Handling:* In [8], Nie, Hannan, and Jin focus on the problem of unknown word detection in Chinese. By first segmenting heuristically, then statistically analyzing unknown strings to determine their likelihood of being real words, they are able to automatically update their dictionary and achieve, at best, a success ratio of 95.66%.
3) *Maximum Entropy:* In [5], Xue approached Chinese word segmentation as a character tagging problem. By training a MaxEnt model using features involving neighboring characters, Xue tags each character in a string by its most likely position; left, right, middle, or alone. From that information, a most likely segmentation is returned, yielding 94.96% f-score.
4) *Conditional Random Fields:* In [9], Peng, Feng, and McCallum, take a similar approach, but use CRFs as opposed to MaxEnt models to tag characters as either START or NONSTART. Using POS tags and neighboring characters as features, as well as an N-best system to process and accept probable unknown words, they achieve as high as 95.7% f-score.

IV. Disambiguation methods

Each of the following methods defines a scoring function whose aim is to assign top scores to correct segmentations, thus turning the process of segmentation into a search for the highest score.

A. Maximum Known Matching (MKM)

To begin, we try a simple maximum matching approach—i.e. given a string s, get all possible segmentations of s into dictionary words, then return the "longest" segmentation.

The question then becomes how to define the length of a segmentation. Should we prefer the segmentation containing the longest words? That which has the largest average word length?

We want to consider both. We first want to consider just the segmentations with the largest average word length, then take that which contains the longest word (if the longest words are equal then compare the second longest, third, etc). In other words we need a function f that fulfills the following two conditions:

1) $size(s_1) > size(s_2) \implies f(s_1) < f(s_2)$
2) $size(s_1) = size(s_2) \wedge s_1 > s_2 \implies f(s_1) > f(s_2)$

Where s_1 and s_2 are segmentations, $size$ a function that returns the number of words in a segmentation, and $s_1 > s_2$ meaning s_1 contains longer words than s_2. (Note that average word length is inversely proportional to the number of words in a segmentation).

We could of course write out the logic above, i.e. define a function that takes the segmentation of shortest length containing the largest individual words, but it may be useful later on to be able to assign a numeric score which models the same choice. Thus, we define the length score of a segmentation as follows:

$$score(s) = \sqrt[i]{\sum_{k=1}^{i} len(w_k)^2}$$

Where $len(w)$ returns the length of a word w, and s is a segmentation into i words.

For example, the score of "bra in storm" would be $\sqrt[3]{3^2+2^2+5^2} \approx 3.36$ whereas the score of "brainstorm" would be $10^2 = 100$. This scoring function indeed satisfies condition 2, and, for the values we are working with, very closely approximates condition 1. Hypothetically, it could fail on say a segmentation of length ten with words of lengths 11,1,1,1,1,1,1,1,1,1, versus a segmentation of length nine with words of lengths 2,2,2,2,2,2,2,3,3, "incorrectly" preferring the former despite the greater average word length of the latter, but segmentations like these are highly improbable and will be easily outscored by less extreme matchings.

B. Maximum Unknown Matching (MUM)

The problem with the previous aptly named approach, however, is that it accepts no unknown words. To amend this, we expand our algorithm in the following way: given a string s, rather than looking at all segmentations into known words, consider all segmentations of s where each division point borders at least one known word. Then return the segmentation with the highest length score.

But now we must amend our definition of length score, for as it stands it will simply return s itself (or, if we exclude s, a segmentation with a very large unknown word). The previous definition of length score has no way of weighing known versus unknown words, and places high value on the average word length of a segmentation. To adjust we redefine the score as follows:

$$score(s) = \frac{\sum_{k=1}^{i} len(known_k)^2 + \sum_{k=1}^{j} len(unknown_k)}{i+j}$$

Where s is a segmentation into i known words and j unknown words.

C. Two grams (2GM)

These simple methods produce fairly effective results (see evaluation section), but, as discussed earlier, successful disambiguation cannot rely merely on length—some syntactic data must be incorporated. Using a database of 2-gram occurences in a corpus, we define the 2-gram score of a segmentation s, simply as the number of recognized 2-grams in s, divided by the total number of 2-grams in s. A segmentation of length 1, thus containing no possible 2-grams, receives a score of -1.

The final scores are then normalized to fall between 0 and 1, and a score of -1 is set by default to 0.5.

We ignore the actual occurence count of the recognized 2-grams in order to avoid over-segmentation. The string "d at a mining", for example, would return a much higher score than the correct "data mining", due to the frequency of the 2-gram "at a", were occurence count taken into consideration. Without it, the latter outscores the former.

To account for numbers and ordinals, which are not included in our 2-gram datasets, as well as acronyms and contractions, we translate each unrecognized word into a set of possible words, via either a translation dictionary—the contents of which are detailed below—or, in the case of numbers and ordinals, a predefined ruleset. Out of these possibile translations, that with the highest 2-gram score is assumed to be correct.

D. POS tagging

2GM is flawed, however, in the same way that MKM is flawed; it lacks strength in handling unknown words. 2-grams which lie outside of the database return a count of 0.

As an attempt to correct this, we approach the problem as a POS tagging problem. I.e. given a segmentation, tag each word with the appropriate POS, then assign it a score based on the probability of a given sequence of POS tags.

This approach poses two new problems:

1) Tag appropriately.
2) Score tag sequences.

And for each we pose two solutions, one using the ARK POS tagger for Twitter [14], and one using Hidden Markov Models (HMM) implemented with the MALLET toolkit [13]. Using ARK, we can tag segmenations by POS, then, using their POS n-gram data, repeat 2GM using POS tags rather than words themselves. Similarly, with a trained HMM, we can tag segmentations by POS, then score a tag sequence by taking its average edge weight in the model. This leads us to four new possible strategies:

1) Tag with ARK, assign probabilities with ARK (AA).
2) Tag with ARK, assign probabilities with HMM (AH).
3) Tag with HMM, assign probabilities with ARK (HA).
4) Tag with HMM, assign probabilities with HMM (HH).

V. Algorithms

We now have seven scoring methods for disambiguation—MKM, MUM, 2GM, plus the four listed above. When used in overall segmentation algorithms, these scores are normalized on each new input to fall between 0 and 1, based on the highest scoring segmentation for the current input.

A. Pipeline

These scores, plus some simple heuristics, leave us with the pipeline-based segmentation algorithm as detailed in the next column.

In the pipeline algorithm, $highestScoringSeg()$ searches by brute force using a scoring function that is some convex combination of previously mentioned scoring functions (MUM, 2GM, AH, etc.); $prune()$ heuristically filters out unlikely segmentations; $allPossibleSegmentations()$ returns every possible segmentation of s—unless s exceeds a length threshold, in which case at least one of the k-longest words in s is made to be present in every segmentation (to limit search space size); and $segmentByDelimeters()$ segments based on punctuation and capitalization, returning, for example "Club Rio - June 19 th - 8 - 12 am" when given "ClubRio-June19th-8-12am". Note that this will fail on confusing inputs. "LadiesoftheDMV", for example, looks delimited but is in fact only partially, and would thus be under-segmented. Only those rule-based segmentations which meet a certain length score threshold, or are composed entirely of known words, therefore, are returned. The rest have their unknown sections fed back into the segmenter to be treated by the normal algorithm.

```
if s is known then
    return s;
end
if s is already delimited then
    return segmentByDelimiters(s);
else
    possible = allPossibleSegmentations(s);
    probable = prune(possible);
    return highestScoringSeg(probable);
end
```

B. Hill-climbing

Based on the work of Zhang et. al. [11], we also consider a greedy hill-climbing algorithm. I.e., rather than pruning down to a set of probable segmentations and then searching by brute force, start with a random segmentation, calculate the scores of all "nearby" segmentations, then climb to the one with the highest score and recurse. The algorithm terminates once no further upward steps are possible. k random restarts are allowed to minimize the chance of getting stuck in local maxima. In our implementation, segmentations of a string of length n string are considered as binary strings of length $n-1$, where a "1" indicates that the corresponding character in the original string is followed by a splitting point. "Nearby" segmentations are then simply defined as the set of binary strings obtained by flipping a bit in the original, i.e. either adding or removing a splitting point.

In its initial implementations, hill-climbing, although more elegant than the pipeline approach, proved to be slightly less successful (at best yielding an f-score of 76.5%, whereas, at the time, the pipeline approach peaked at 82.2%). Likely, the definition of nearby segmentations was too narrow, creating too many local maxima to avoid. Due to time constraints, however, the hill-climbing approach was dropped in order to devote more time towards improving the pipeline method.

VI. Unknown handling

In [8], Nie et. al. define the following process for unknown handling:

1) Perform a maximum matching.
2) Gather remaining unknowns.

3) Remove unlikely candidates based on a predefined ruleset.
4) Add those which occur most frequently to the dictionary and repeat.

This method enables the segmenter to improve with repeated applications, and it is, generally, the method we use to accomodate unknown words. Rather than defining a ruleset for how words should look, however, we train a Markov chain on a list of English words. The states of the chain correspond letters of the alphabet, and transitions between states model letter sequence probabilities. The average transition probability of a given string should, then, theoretically mirror its likelihood of being a real word. The removal of unlikely candidates then comes down to choosing a threshold value. As with Nie's method, the frequency of a given unknown is also factored into its estimated probability of being a real word. Because of this, the size of the test set will directly affect the appropriate threshold value.

In addition to this algorithm, unknown handling has also been attempted via the use of spell-checking resources. Spelling errors are the root cause of many unknowns, and handling them effectively would have significant effects on performance. Several strategies were tested—treating unknown words within some edit distance threshold of known words as known words themselves; treating such words as "semi-known" words, and adjusting the length score function to handle them; including spell-check suggestions as translations for 2GM scoring—but each led to disjunctive ambiguity errors; words would be segmented with extra letters when they shouldn't have. Rules were devised as an attempt to exclude such occurrences, but still without improving results. The one method which did prove to be useful was the inclusion of common misspellings and their root words in our translation dictionary. As with numbers, ordinals, acronyms, and contractions, common spelling corrections have thus been included in translations for 2GM scoring.

VII. Data

The dictionary used for matching includes

- ∼100,000 english words,
- ∼4,000 abbreviations,
- ∼200 slang words,
- ∼330 corporations,
- ∼24,000 names (both first and last), and
- ∼18,000 place names.

The dictionary used for translation includes

- ∼500 acronyms,
- ∼100 contractions, and
- ∼5,000 common misspellings.

All dictionary entries were collected from freely available online resources. For testing, ∼400,000 hashtags were pulled from Rovereto's Twitter 1-gram data [17]. 2-gram data contains the 1,000,000 most common English 2-grams based on the 450 million word Corpus of Contemporary American English [18]. HMMs used in conjunction with the ARK POS tagger were trained on 547 POS tagged tweets (7707 tagged words) from data used by ARK [14]. HMMs used alone were tried first on the ARK dataset, then on a manually curated supervised dataset of 1,000 hashtags using the following alternative tagset:

- NO: noun
- VE: verb
- AD: adjective
- DE: determiner
- PR: preposition
- CO: conjunction
- NU: number
- OR: ordinal
- AB: abbreviation
- FN: first name
- LN: last name
- PL: place
- MO: month
- DA: day of the week
- TE: Twitter team (e.g. "Team Iphone")
- TI: title (e.g. "mr", "mrs", "lord")
- PU: punctuation
- O: other

Dictionaries of names and places were also added as length-1 hashtags to increase word recognition by the HMM. The HH method using this training proved more effective than using ARK data, though still not up to par with the 2GM method. In the end, due to time constraints, HH work, as with work on the hill-climbing algorithm, was stymied in order to focus on maximizing 2GM results.

An attempt was also made to include a lexical normalization dictionary, taken from Han, Cook, and Baldwin's research on normalization for microblogs [12], as part of the translation dictionary, but the data proved too noisy to be useful.

VIII. Evaluation

Performance is rated in terms of precision, recall, and f-score. For a single segmentation, precision is defined as the number of correctly segmented words divided by the total number of words *in the proposed segmentation*, and recall as the number of correctly segmented words divided by the total number of words *in the correct segmentation*. This leads to overall precision and recall scores for a list of k hashtags defined simply as the average scores for all segmentations, i.e.

$$P = \frac{1}{k} \sum_{i=1}^{k} p_k$$

$$R = \frac{1}{k} \sum_{i=1}^{k} r_k$$

And f-score is calculated as the harmonic mean of the two:

$$F = \frac{2PR}{P+R}$$

[6]. Additionally, two new metrics are provided, *known-precision* and *known-recall*. These are simply precision and recall measured in terms of *known* words in a segmentation, rather than *all* words in a segmentation. Generally, known-precision scores are higher than precision scores, and known-recall lower than recall. The purpose of these measures is meant to be more pragmatic, e.g. someone requiring higher

precision at the expense of lower recall could choose to only consider the known words in a proposed segmentation.

Correct segmentations are based on a manually segmented list of 1129 hashtags. In the list, each hashtag corresponds to a set of all valid segmentations (typically just one, but in some cases alternate segmentations exist which are equally acceptable). During calculation of evaluation metrics on a proposed segmentation, scores are calculated for each possible answer and the highest results are returned.

The following table lists the results of methods that have been tested with the current dictionaries and heuristics, as tested on our manually curated answer set.

Method	Prec	Rec	F-score	KPrec	KRec
MKM	0.901	0.909	0.905	0.912	0.835
MUM	0.916	0.918	0.917	0.935	0.825
2GM	0.921	0.924	0.923	0.942	0.829
AH	0.921	0.923	0.922	0.938	0.830

In preliminary testing AH outscored the other three POS-based methods. This makes sense, as ARK's successful POS tagger should easily trump a our HMMs in terms of tagging accuracy, whereas the edge weights of the HMM should provide comparable or better transition probabilities. As such, AH was the first (and as of yet, only) of the four to be tested with the updated heuristics and newly introduced translation scheme.

For efficiency, possible segmentations had to be pruned twice before actually taking AH score into consideration. First, heuristically. Second, by a combination of length and 2GM score. This second pruning was based on an optimal convex combination of length and 2GM scores, the values of which are depicted in figure 1. The remaining segmentations were then disambiguated by a convex combination of all three scores, length, 2-gram, and AH. Figure 2 displays the relationship between possible combinations of the three and their effects on f-score. Although the success of the AH method is largely due to the success of the pipeline system, figure 2 shows that it can perform as well as the 2GM method in disambiguation.

In figue 1, L refers to the weight assigned to the length score, and T, the weight assigned to the 2GM score, can be simply calculated as 1-L. Similarly, in figure 2, L represents the weight of the length score, A the weight of the AH score, and T is left to be calculated as 1-(L+A). Performance evaluation in figure 2 is based on the segmentation of 232 hashtags chosen such that they could not be handled by simple heuristics. Physically, they were chosen by running a heuristic segmenter on the manually curated collection and gathering those on which the machine failed. Performance in figure 1 is evaluated based on the full answer set.

IX. Possible Improvements

Areas to be improved upon are varied. With the right tagset and enough supervised learning data, there is still hope for the success of the HH method. Alternatively, CRFs, as shown in [9], have been used successfully as segmentation tools, and moving from HMMs to CRFs, thus allowing arbitrary

Fig. 1.

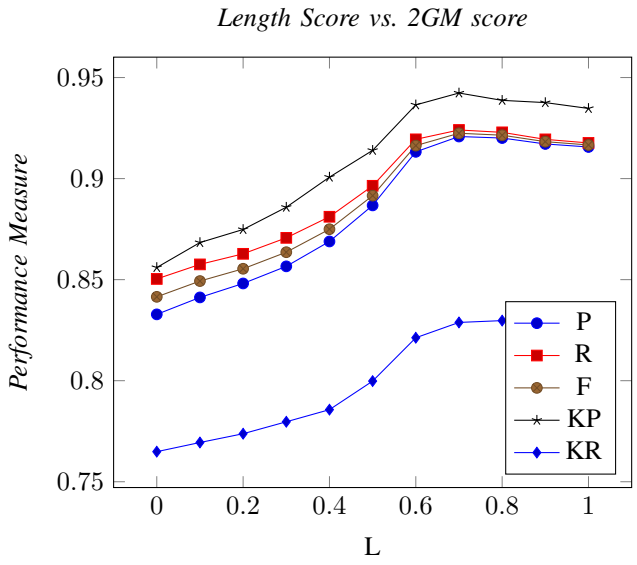
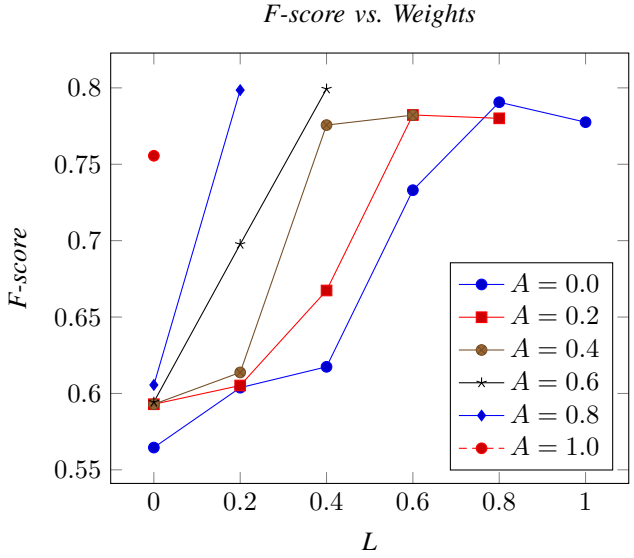


Fig. 2.



feature inclusion, could be a more effective option. Larger n-gram data has not yet been tried—3-grams, 4-grams, etc.—to extend 2GM, and neither have alternative lexical normalization strategies. With different definitions of distance, the hill-climbing algorithm could also prove superior to the pipeline approach.

Much work is also left to be done in the task of unknown handling. In our research, the character HMM method has not been extensively developed. Different training sets could be explored, as well as more sophisticated methods for learning threshold values. Additionally, as far as spelling correction goes, our solution is far from perfect. A distance measure that balances spelling error correction against the creation of erroneous disjunctive ambiguities would likely improve performance greatly. Physical keyboard proximity may even be useful to consider.

X. Conclusion

Hashtags typify a significant chunk of conversational language online. They have spread beyond Twitter and into most popular social media sites. This project provides the foundations of a tool to accurately segment hashtags into meaningful subdivisions. This will hopefully extend the ability of machines to understand the enormous amount of data produced on the Internet constantly, primarily by social and other informal media outlets.

One immediate extension of such a tool would be to use a resource like WordNet to create a graph of relationships among hashtags, allowing machines to first process a hashtag, then not only explore related topics within that tag, but other topics within related tags. Relationships could also be estimated by viewing hashtags from the "bag of words" perspective, and then applying machine learning techniques such as clustering to group them appropriately.

The same segmentation methods could also be applied to similar strings. Urls, for instance, are the first that come to mind.

References

[1] Michael J. Paul and Mark Dredze, "You Are What You Tweet: Analyzing Twitter for Public Health", *Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media*, Association for the Advancement of Artificial Intelligence, 2011.

[2] Michael D. Conover et. al., "Predicting the Political Alignment of Twitter Users", *IEEE International Conference on Privacy, Security, Risk, and Trust, and IEEE International Conference on Social Computing*, pp. 192-199, 2011.

[3] Hemant Purohit, "What kind of #conversation is Twitter? Mining #psycholinguistic cues for emergency coordination", *Computers in Human Behavior*, Vol. 29, Issue 6, pp. 2438-2447, 2013.

[4] Jonathan J. Webster and Chunyu Kit, "Tokenization as the initial phase in NLP", *Proceedings of the 14th conference on Computational linguistics*, Vol. 4, pp. 1106-1110, Association for Computational Linguistics Stroudsburg, PA, USA, 1992.

[5] Ninanwen Xue, "Chinese Word Segmentation as Character Tagging", *Computational Linguistics and Chinese Language Processing*, Vol. 8, No.1, pp. 29-48, 2003.

[6] Sebastian Spiegler and Christian Monson, "EMMA: A Novel Evaluation Metric for Morphological Analysis", *Proceedings of the 23rd International Conference on Computational Linguistics*, pp. 1029-1037, 2010.

[7] Phillipp Koehn and Kevin Knight, "Empirical methods for compound splitting", *EACL '03 Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics*, Vol 1, pp. 187-193, Association for Computational Linguistics Stroudsburg, PA, USA, 2003.

[8] Jian-Yun Nie, Marie-Louise Hannan, and Wanying Jin, "Unknown word detection and segmentation of chinese using statistical and heuristic knowledge", *Communications of COLIPS 5.1*, pp. 47-57, 1995.

[9] Fuchun Peng, Fangfang Feng, and Andrew McCallum, "Chinese segmentation and new word detection using conditional random fields", *COLING '04 Proceedings of the 20th international conference on Computational Linguistics*, Artical No. 562, Association for Computational Linguistics Stroudsburg, PA, USA, 2004.

[10] Hoifung Poon, Colin Cherry, and Kristina Toutanova, "Unsupervised morphological segmentation with log-linear models", *NAACL '09 Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 209-217, Assocation for Computational Linguistics, Stroudsburg, PA, USA, 2009.

[11] Yuan Zhang, Chengtao Li, Regina Barzilay, and Kareem Darwish, "Randomized Greedy Inference for Joint Segmentation, POS Tagging and Dependency Parsing", *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computations Linguistics: Human Language Technologies* pp. 42-52, Association for Computational Linguistics, Denver, CO, 2015.

[12] Bo Han, Paul Cook, and Timothy Baldwin, "Automatically constructing a normalisation dictionary for microblogs", *Proceedings of EMNLP-CoNLL 2012*, pp 421-432, Kora, 2012.

[13] Andrew Kachites McCallum, "MALLET: A Machine Learning for Language Toolkit", "http://mallet.cs.umass.edu", 2002.

[14] Olutobi Owoputi, et. al., "Improved part-of-speech tagging for online conversational text with word clusters", *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 380-391, Association for Computational Linguistics, 2013.

[15] John Goldsmith et. al., "The Linguistica Project", "http://linguistica.uchicago.edu".

[16] Alec Go, Richa Bhayani, and Lei Huang, "Twitter sentiment classification using distant supervision", *CS224N Project Report, Stanford*, Vol. 1, pp. 1-12, 2009.

[17] Ama Herdadelen, "Rovereto Twitter N-Gram Corpus: An n-gram dataset of Twitter messages with gender labels and time of posting", http://clic.cimec.unitn.it/amac/twitter_ngram/,

[18] "N-grams data", http://www.ngrams.info/