SEGMENTING TWITTER HASHTAGS

Jack Reuter [1], Jhonata Pereira-Martins[1] and Jugal Kalita[2]

[1]Department of Computer Science, Wesleyan University Middletown CT 06459
[2]Department of Computer Science, University of Colorado Colorado Springs CO 80918

ABSTRACT

Social Media Posts On Platforms Such As Twitter Or Instagram Use Hashtags, Which Are Author-Created Labels Representing Topics Or Themes, Toassist In Categorization Of Posts And Searches For Posts Of Interest. The Structural Analysis Of Hashtags Is Necessary As Precursor To Understandingtheir Meanings. This Paper Describes Our Work On Segmenting Nondelimited Strings Of Hashtag-Type English Text. We Adapt And Extend Methods Used Mostly In Non-English Context For Word Segmentation, To Handle Hashtags In English, Obtaining Effective Results.

KEYWORDS

Hashtags, segmentation, Twitter

1. INTRODUCTION

Social media sites such as Twitter use hashtags, which are key phrases preceded by the pound (#) symbol to mark posts with similar themes. A hashtag can be placed anywhere within a tweet, and it is usually used to highlight the tweet's keywords or main subject matter. Popular hashtags are presented on Twitter as trending topics. The discussion of celebrities, TV shows and movies, natural disasters like fires and earthquakes, and political hot topics all use hashtags so that, when searching a site like Twitter, posts on the topic of interest can be easily found. From these hashtags, Twitter also compiles lists of the most noteworthy topics of the day, month, year.

On Twitter, a post may contain none, one, or as many hashtags as the user may desire (or may fit within the 140 character limit). Research has shown that tweets with hashtags receive two times the engagement than those without, and that tweets with one or two hashtags have 21% higher engagement than those with three or more1. In addition to Twitter, hashtags are used copiously in other social media sites such as Instagram, Pinterest, Google+, Tumblr and Orkut. Starting in 2013, the social media giant, Facebook, has supported Twitter-like hashtags2. Consulting about how to use hashtags properly as a marketing tool, and analysis of hashtags to track and measure hashtag engagement and influence have spawned many businesses, including websites such as hashtags.org.

A hashtag is simply a key phrase, composed of one or more tokens, written without whitespace. Although the tokens are frequently words of a language like English, they do not have to be. One can use abbreviations, wrongly spelled words, made-up words containing arbitrary character sequences, numeric digits, names of people, places or events; in short, anything possible on an everyday keyboard. One needs to be mindful about keeping a hashtag short, for both readability and for actual length limitations imposed by previously mentioned sites. In Table 1 we see some "English" hashtags that are composed of English or English-like tokens. Of course, it is possible to mix languages, such as English and Spanish, or English and German in the same hashtag.

DOI: 10.5121/ijnlc.2016.5402

Hashtag	Preferred tokenization	Alternate tokenization
#wordsoftheday	words of the day	word soft he day?
#statefarmisthere	state farm is here	state far mist here?
#brainstorm	brainstorm	bra in storm?
#doubledown	double down	do u bled own?
#votedems	vote dems	voted ems?

Table 1: A few hashtags, showing examples of ambiguous tokenizations, along with preferred tokenizations.

Word segmentation is an important first step in natural language processing. It is difficult to derive meaning from a piece of text without first having a good understanding of the words that it comprises. We believe that hashtag tokenization should be the first step in automatic understanding, clustering or classification not only of individual hashtags, but of social media posts containing them. In this paper, we discuss methods for, experiments in, and results of automatic hashtag segmentation.

2. PROBLEM DEFINITION

Formally, the problem is as follows: Given a string a1a2:::ak, where each ai is a meaningful substring (i.e., a word, name, abbreviation, etc.), determine where the boundaries lie between each ai and ai+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.” “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” [1]. 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, or has 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”. Following Webster and Kit [2], 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- match able 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 our effort to succeed, such unknowns must be recognized and handled appropriately.

3. RELATED WORK

Twitter posts are voluminous in number, publicly available and easily accessed, and as a result, they have become a prime source of "modern" Internet-mediated language for studies of all stripes, with hundreds of papers published, including a recent survey [3]. For example, studies have attempted to normalize the informality present in such language [4, 5, 6], tag parts-of-speech [7, 8], summarize a collection of posts [9, 10, 11], identify topics [12, 13] and detect expressed sentiments [14, 15]. To the best of our knowledge, however, most of these studies simply ignore hashtags, which are crucial for indexing and categorizing tweets.

In languages such as German and Chinese, due to frequent use of large compounds and lack of delimitation, respectively, effective segmentation is important. Nie et al. [16] segment Chinese text heuristically, analyze unknown strings statistically for likelihood of being real words. Xue et al. [1] segment Chinese words by tagging them with a MaxEnt model trained using features involving neighboring characters. Peng et al. [17] use Conditional Random Fields (CRFs) 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. Koehn and Knight [18] learn splitting rules for German compounds and choose most likely segmentations based on how frequently the segmented tokens appear in a corpus, as well as whether or not the translation of a segmentation contains the same words as the translation of the original string. Part-of-speech (POS) tags are also used to help avoid splitting off suffixes and prefixes from root lemmas.

There is not much prior work in hashtag segmentation. Berardi et al. [19], who took part in TREC Microblog Track 2011 [20] for searching a set of 16 million tweets for relevant tweets at a certain time, discuss approaching hashtag segmentation using a Viterbi-type algorithm [21] to improve search performance, although they do not provide any results. Srinivasan et al. [22] discuss several approaches primarily devoted to segmentation of web domain names, although they discuss the segmentation of hashtags briefly. Their contention is that domain names and hashtags share certain similar properties in how they are constructed and therefore, similar approaches can be used. They start with several corpora, and develop scoring formulas for segmentation using unigrams and bigrams found in the corpora and segmentation length. Models obtained are then weighed using a maximum margin learning problem solved using the SVM- struct framework [23]. They do not provide any direct results for hashtag segmentation, but do show that it can improve recall in Twitter searches. Bansal et al. [24] show that hashtag segmentation can improve linking of entities extracted from tweets. They assume that an unknown or OOV segment inside a hashtag is surrounded by known segments on its two sides. If there are ambiguities, they hypothesize various lengths for the intervening segment, scoring the alternatives using unigram based features and probable lengths of the unknown segment. Probable lengths are determined assuming its length follows the distribution of lengths of known tokens found in tweets.

Segmentation of a hashtag involves several steps, as we will soon see. The current wisdom is that when there is a pipeline involving several steps of processing in an NLP task, the results can often be improved by dispensing with pipelining and performing the steps jointly. For example,

Zhang et al. [25] improve performance of a greedy algorithm that jointly predicts segmentation, POS tags and dependency parses, improving all of them iteratively, using hill-climbing. The contention is that in a pipeline architecture, the errors cascade from one step to the next, but joint prediction reduces error propagation. Joint prediction [26, 27, 28] has performed better compared to pipeline architectures, especially for morphologically rich languages like Arabic and Chinese.

4. OUR APPROACH

In the rest of the paper, we describe the several approaches we take to hashtag segmentation and present our results. We first present the methods we use to generate and assign scores to possible segmentations for a given input. We then present the more general algorithm we use to select the best answer from the generated set of possibilities. After implementing the algorithm's steps in pipeline, we also describe a hill-climbing version of the algorithm, because we want to investigate if joint prediction performs better for the hashtag segmentation task, as claimed by several recent papers in similar tasks. We finally present segmentation results produced by our approaches, compare them, and identify the best performer.

5. SEGMENTATION SCORING 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. We have developed these scoring functions based on our understanding of what ideal segmentations should look like. The goal is to give high scores to good segmentations and low scores to bad ones.

5.1 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(s1) > size(s2) =) f(s1) < f(s2)
2. size(s1) = size(s2) ^ s1 > s2 =) f(s1) > f(s2)

where s1 and s2 are segmentations, size a function that returns the number of words in a segmentation, and s1 > s2 meaning s1 contains longer words than s2. 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.

5.2 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.

5.3 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 occurrences 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.

5.4 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: a) Tag appropriately, and b) Score tag sequences. And for each we pose two solutions, one using the ARK POS tagger for Twitter [8], and one using Hidden Markov Models (HMM) implemented with the MALLET toolkit [29]. 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), and
4. Tag with HMM, assign probabilities with HMM (HH).

6. 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.

6.1 Pipeline

These scores, plus some simple heuristics, leave us with the pipeline-based segmentation algorithm as detailed in Algorithm 1.

Algorithm 1 Hashtag Segmentation Algorithm

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

In the pipeline algorithm, *highest Scoring Seg* () 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*; all Possible Segmentations* () 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 *segment By Delimeters* () 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.

6.2 Hill-climbing

Based on the work of Zhang et al. [25], 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.

7. UNKNOWN HANDLING

Nie et al. [16] 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 to letters of the

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")	PU: punctuation
TI: title (e.g. "mr", "mrs", "lord")	O: other

Table 2: POS tags used

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, intentional or unintentional, 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.

8. RESOURCES USED

We use a number of dictionaries for matching. The dictionaries used include ∼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 dictionaries 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 [30], 1129 of which were manually segmented to test against. 2-gram data contains the 1,000,000 most common English 2-grams based on the 450 million word Corpus of Contemporary American English[3]. HMMs used in conjunction with the ARK POS tagger were trained on 547 POS tagged tweets (7707 tagged words) from data used by ARK [8]. HMMs used alone were tried first on the ARK dataset, then on a manually curated supervised dataset of 1,000 hashtags using the alternative tagset given in Table 2.

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. An attempt was also made to include a lexical normalization dictionary, taken from [6], as part of the translation dictionary, but the data proved too noisy to be useful.

9. EVALUATION AND RESULTS

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*

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
HH	0.919	0.922	0.921	0.940	0.829

Table 3: Performance evaluation in terms of accuracy

Scoring Function	Hashtags/Second
Length	330,000
2GM	21,000
AH	0.65
HH	23

Table 4: Performance evaluation in terms of speed

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: $F = \frac{2PR}{P+R}$ [31]. 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.

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.

Tables 3 and 4 list the results of methods that have been tested with the current dictionaries and heuristics, as tested on our manually curated answer set. AH outscored the other three POS-based methods. This makes sense, as ARK's successful POS tagger should easily trump 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 of the four to be tested with the updated heuristics and newly introduced translation scheme. AH was followed by HH for the latter's far superior speed.

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.

The same pruning system was used for the HH method, though with a slightly larger pruning set size. Figure 3 mirrors the form of Figure 2, with HH score substitued for AH score. Though faster than AH, HH has been less successful in disambiguation. In Figure 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 Figures 2 and 3, L represents the weight of the length score, and T is left to be calculated as 1-(L+A) or 1-(L+H), respectively, where A represents the weight of the AH score and H the HH score. Performance evaluation in Figures 2 and 3 is based on the segmentation of 232 hashtags chosen such that they could not be handled by simple heuristics. 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.

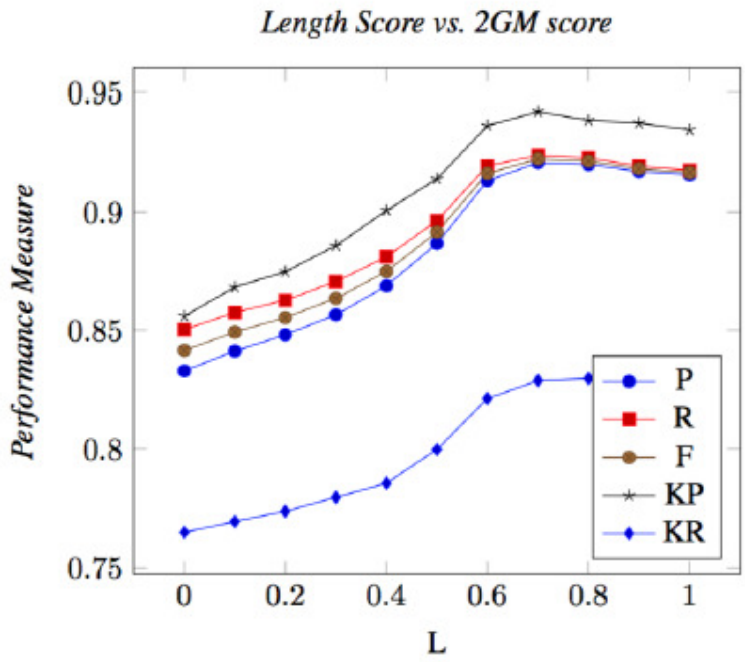

Figure 1: A range of combinations of length score and 2GM score with resulting performance measures

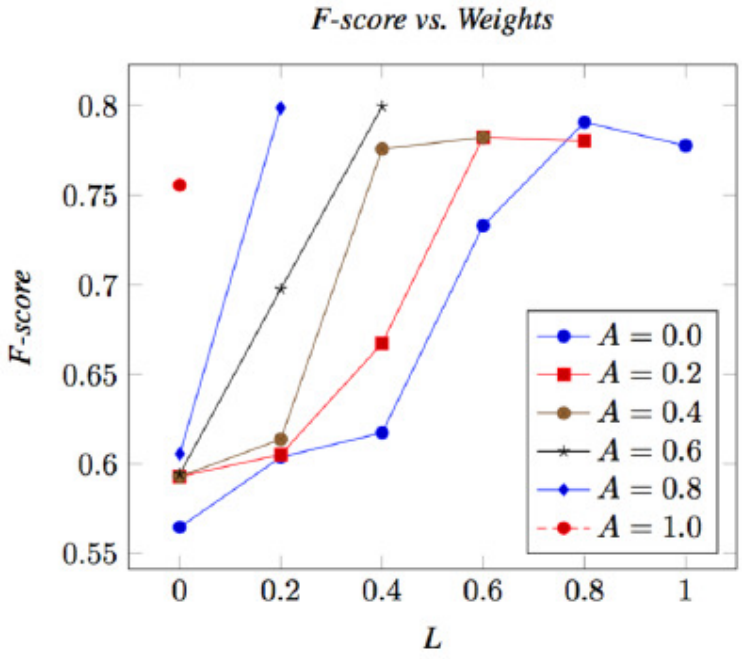

Figure 2: A range of combinations of length score, 2GM score, and AH score with resulting performance measures

9.1 Comparison with Prior Approaches

Srinivasan et al. [22] do not provide direct results for hashtag segmentation, but state that recall in search for tweets can improve by 5% or more when they segment the hashtags and use the tokens obtained in search. They developed various models for segmentation of domain names, and obtained average accuracies of 87.07, 87.98, 88.68 and 89.74% for the four bigram-based methods (these numbers were obtained by adding up the values in the bottom part of Table 3 in their paper and averaging). Our methods perform better compared to their average results, though their numbers are for domain splitting and ours for hashtag splitting. The work that comes closest to ours is by Bansal et al. [24], who segment Twitter hashtags with the goal of improving extraction of links or relations among entities mentioned in tweets. They provide

Method	Precision	Recall	F-measure
MKM	0.762	0.781	0.771
MUM	0.804	0.773	0.788
2GM	0.857	0.824	0.840

Table 5: Performance evaluation in terms of accuracy for Brazilian Portuguese

a P@1 or accuracy value (which is the precision value we report) as 0.914. This is quite comparable to the results we provide, but they do not provide any recall and F-score values. Hence, it is difficult to fully compare their method to ours.

10. EXPERIMENTS WITH BRAZILIAN PORTUGUESE

We wanted to experiment with hashatags in at least one more language and we chose it to be Brazilian Portuguese. Although often entirely English, tweets in Brazilian Portuguese range from being entirely English to entirely Portuguese, sometimes combining both. Of course, working with both languages brings in more difficulties, mainly more ambiguities.

Since these two languages are not very dissimilar, we hypothesized that many ambiguities could be handled by methods previously described for English. We kept the original English dictionary with approximately 150,000 words, but added to it a dictionary of approximately 100,000 Portuguese words obtained from free online resources. We added abbreviations, corporations, first and last names to the Portuguese dictionary, just as we did for English. We also added the 50,000 most common 2-grams in Portuguese taken from Corpus do Portuguˆes, and downloaded ~100,000 hashtags in Brazilian Portuguese from Twitter using the Twitter API.

Although formal Brazilian Portuguese uses accents liberally, accents are often omitted in tweets due to typing ease on mobile devices. To simplify matters, we removed all accents from our datasets. For experiments reported here, we added Portuguese 2-gram data to our English 2-gram data, although we acknowledge that this misses the possibility of mixed English and Portuguese 2-grams.

Correct segmentations are based on randomly chosen 1000 manually segmented hashtags. Results from Brazilian Portuguese with MKM, MUM and 2GM methods are given in Table 5. Additional experiments were not performed for Portuguese. We believe that the combination of two languages and lack of as many resources as in English has for ready use, caused the results in Brazilian Portuguese to be poorer.

11. POSSIBLE IMPROVEMENTS AND CONCLUSION

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 [17], have been used successfully as segmentation tools, and moving from HMMs to CRFs, thus allowing arbitrary feature inclusion, could be a more effective option. Larger n-gram data has not yet been tried 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. Different training sets could be explored, as well as sophisticated methods for learning threshold values. 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.

Hashtags typify a significant chunk of conversational language online. They have spread beyond Twitter and into most popular social media sites. Some Twitter posts contain two, three or more hashtags. Instagram posts can contain ten or more hashtags. Thus, ignoring hashtags in analyzing texts in social media posts can make for impoverished understanding. One extension could be to create a graph of relationships among

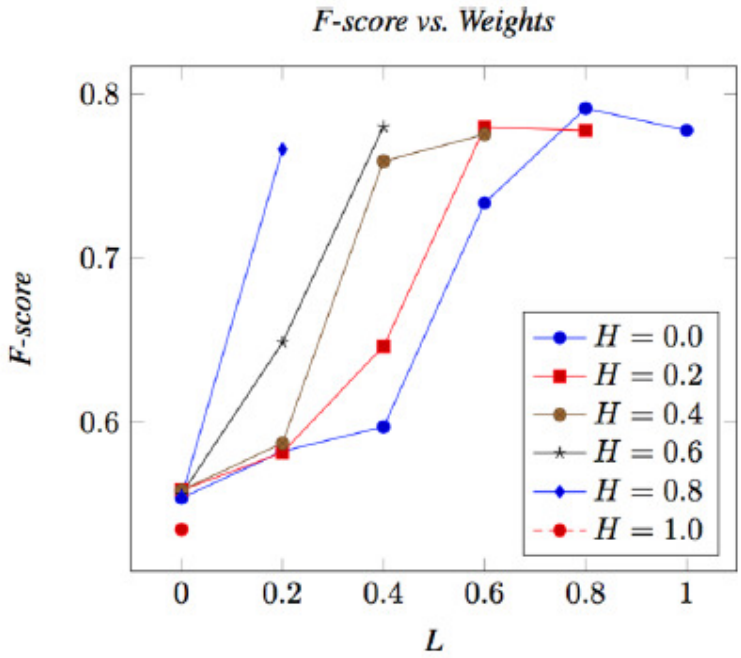

Figure 3: A range of combinations of length score, 2GM score, and HH score with resulting performance measures

hashtags, allowing machines to first process a hashtag, then not only explore related topics within that tag, but other topics within related tags.

REFERENCES

[1] N. Xue, "Chinese word segmentation as character tagging," Computational Linguistics and Chinese Language Processing, vol. 8, no. 1, pp. 29–48, 2003.

[2] J. J. Webster and C. Kit, "Tokenization as the initial phase in Natural Language Processing," in COLING–Volume 4, 1992, pp. 1106–1110.

[3] A. Farzindar and D. Inkpen, "Natural language processing for social media," Synthesis Lectures on Human Language Technologies, vol. 8, no. 2, pp. 1–166, 2015.

[4] M. Kaufmann and J. Kalita, "Syntactic normalization of twitter messages," in International Conference on Natural Language Processing, ICON, Kharagpur, India, 2010.

[5] B. Han and T. Baldwin, "Lexical normalisation of short text messages: Makn sens a# twitter," in ACL-HLT, Volume 1, 2011, pp. 368–378.

[6] B. Han, P. Cook, and T. Baldwin, "Automatically constructing a normalisation dictionary for microblogs," in EMNLP and CoNLL, 2012, pp. 421–432.

[7] K. Gimpel, N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yogatama, J. Flanigan, and N. A. Smith, "Part-of-speech tagging for twitter: Annotation, features, and experiments,"in ACL-HLT, Volume 2, 2011, pp. 42–47.

[8] O. Owoputi, B. O'Connor, C. Dyer, K. Gimpel, N. Schneider, and N. A. Smith, "Improved part-ofspeech tagging for online conversational text with word clusters," in NAACL-HLT, 2013, pp. 380–390.

[9] B. Sharifi, M.-A. Hutton, and J. Kalita, "Summarizing microblogs automatically," in NAACL, 2010, pp. 685–688.

[10] D. Inouye and J. K. Kalita, "Comparing twitter summarization algorithms for multiple post summaries," in Third IEEE International Conference on Social Computing, SocialCom. IEEE, 2011, pp.298–306.

[11] J. Nichols, J. Mahmud, and C. Drews, "Summarizing sporting events using twitter," in ACM International Conference on Intelligent User Interfaces, 2012, pp. 189–198.

[12] M. Cataldi, L. Di Caro, and C. Schifanella, "Emerging topic detection on twitter based on temporal and social terms evaluation," in Proceedings of the Tenth International Workshop on Multimedia Data Mining, 2010, p. 4.

[13] W. X. Zhao, J. Jiang, J. He, Y. Song, P. Achananuparp, E.-P. Lim, and X. Li, "Topical keyphrase extraction from twitter," in NAACL-HLT–Volume 1, 2011, pp. 379–388.

[14] A. Go, R. Bhayani, and L. Huang, "Twitter sentiment classification using distant supervision," CS224N Project Report, Stanford, vol. 1, p. 12, 2009.

[15] L. Barbosa and J. Feng, "Robust sentiment detection on twitter from biased and noisy data," in COLING: Posters. Association for Computational Linguistics, 2010, pp. 36–44.

[16] J.-Y. Nie, M.-L. Hannan, and W. Jin, "Unknown word detection and segmentation of Chinese using statistical and heuristic knowledge," Communications of COLIPS, vol. 5, no. 1, pp. 47–57, 1995.

[17] F. Peng, F. Feng, and A. McCallum, "Chinese segmentation and new word detection using conditional random fields," in COLING, 2004, p. 562.

[18] P. Koehn and K. Knight, "Empirical methods for compound splitting," in European ACL–Volume 1, 2003, pp. 187–193.

[19] G. Berardi, A. Esuli, D. Marcheggiani, and F. Sebastiani, "ISTI@ TREC Microblog Track 2011: Exploring the Use of Hashtag Segmentation and Text Quality Ranking," in TREC, 2011.

[20] I. Ounis, C. Macdonald, J. Lin, and I. Soboroff, "Overview of the TREC-2011 Microblog Track," in Proceedings of the 20th Text REtrieval Conference (TREC 2011), 2011.

[21] G. D. Forney Jr, "The Viterbi algorithm," Proceedings of the IEEE, vol. 61, no. 3, pp. 268–278, 1973.

[22] S. Srinivasan, S. Bhattacharya, and R. Chakraborty, "Segmenting web-domains and hashtags using length specific models," in Proceedings of the 21st ACM International Conference on Information and Knowledge Management, 2012, pp. 1113–1122.

[23] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun, "Support vector machine learning for interdependent and structured output spaces," in Proceedings of the Twenty-first International Conference on Machine Learning. ACM, 2004, p. 104.

[24] P. Bansal, R. Bansal, and V. Varma, "Towards deep semantic analysis of hashtags," in Advances in Information Retrieval. Springer, 2015, pp. 453–464.

[25] Y. Zhang, C. Li, R. Barzilay, and K. Darwish, "Randomized Greedy Inference for Joint Segmentation, POS Tagging and Dependency Parsing," in NAACL-HLT, 2015, pp. 42–52.
[26] N. Habash and O. Rambow, "Arabic tokenization, part-of-speech tagging and morphological disambiguation in one fell swoop," in Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, 2005, pp. 573–580.
[27] `J. Hatori, T. Matsuzaki, Y. Miyao, and J. Tsujii, "Incremental joint approach to word segmentation, pos tagging, and dependency parsing in chinese," in Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1, 2012, pp. 1045–1053.
[28] B. Bohnet, J. Nivre, I. Boguslavsky, R. Farkas, F. Ginter, and J. Haji˘c, "Joint morphological and syntactic analysis for richly inflected languages," Transactions of the Association for Computational Linguistics, vol. 1, pp. 415–428, 2013.