A Project Report

On

Segmentation-Based Event Detection

BY

Keval Morabia

2015A7PS0143H

Under the supervision of

Prof. Aruna Malapati

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF

CS F377: DESIGN ORIENTED PROJECT

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

HYDERABAD CAMPUS

(April 2018)

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to all those who provided me with the possibility to complete this report. Many thanks go to the head of the project, Prof. Aruna Malapati who has invested her full effort in guiding me in achieving the goal. I have to appreciate the guidance given by other supervisors Prof. N.L. Bhanu Murthy and Mr. Surender Singh Samant, especially in the project presentation that has pointed out some of the difficulties with my project and provided solutions for those problems.

Birla Institute of Technology and Science-Pilani,

Hyderabad Campus

Certificate

This is to certify that the project report entitled "**Segmentation-Based Event Detection**" submitted by Mr. Keval Morabia (ID No. 2015A7PS0143H) in partial fulfillment of the requirements of the course CS F377, Design Oriented Project Course, embodies the work done by him under my supervision and guidance.

Date: 26/11/2018 **(Prof. Aruna Malapati)**

BITS- Pilani, Hyderabad Campus

ABSTRACT

Event Detection has been one of the research areas in Natural Language Processing (NLP) that has attracted attention during this decade due to the widespread availability of social media data specifically twitter data. Twitter has become a major source for information about real-world events. This report mentions the problems associated with event detection from tweets and a segment-based method for event detection. The main idea is to split each tweet into non-overlapping segments, extract bursty segments and cluster segments.

Keywords: Event detection, Twitter, Microblogging, Tweet segmentation, N-gram, Wikipedia

CONTENTS

1. Introduction

Microblogging, as a form of social media, is fast emerging in this last decade. One of the best examples for this is Twitter which allows 140-character limit (280 for some people) for each tweet. It is used not only to share and communicate with friends and family but also as a medium to share real-world events. Each user not only can publish about an event but also can propagate by retweeting the post by someone else. But event detection from tweets faces many challenges like short and noisy data, diverse and fast-changing events and a large volume of data. According to "Internet Live Status", on an average 6,000 tweets are published every second, which corresponds to nearly 500 million tweets per day. Moreover, most of the tweets are just pointless "babbles" which are insignificant to the task of event detection. This report presents a segment-based event detection technique which is referred from the research paper – Twevent: Segment-based Event Detection from Tweets [1].

2. Preprocessing

The data obtained from Twitter API has a lot of attributes like **created_at** (e.g. Sun Sep 10 06:30:00 +0000 2017), **id** which is unique for each tweet, **text** which is the content of the tweet, **in_reply_to_status_id** which is either null or a tweet id in which case it means that the data is not a tweet but a comment on someone else's tweet, **user:id** which is unique for each user, **user:name**, **user:location**, **user:followers_count, user:lang, geo** which is the geolocation associated with the tweet, **lang** which is the languages in which the tweet is posted, retweet**_count**, **hashtags, links,** etc. Out of all these attributes, only a few are required in this task which is created_at, text, and user: id. So preprocessing is required for Twitter data.

Twitter data removed:

- Tweets that have in_reply_to_status_id not null
- Tweets with language, not English
- Data about deleted tweets
- Non-ASCII characters from text like emojis
- Hyperlinks in text
- Hashtags and name mention removed from text but present in entities
- Retweets

Twitter data kept after preprocessing:

- created_at
- text in lowercase after splitting compound words, removing stopwords and words of unit length
- user id, followers count
- Hashtags and name mention in entities
- retweet_count

Sample tweet data in JSON format before and after preprocessing is shown in Appendix A-1

3. Segment Based Event Detection

3.1 Overview

As the twitter data keeps on growing as new tweets are posted, it becomes almost impossible to analyze all the tweets together for detecting events. So, the concept of the time window is used. In this, events are continuously detected only from the tweets within a certain time interval called the length of the time window. A time window is divided into ‘m‘ sub-windows of time interval ‘t’ each. So, the entire time window contains tweets from last ‘t*m’ time interval. After each ‘t’ time interval, the time window advances discarding the first subwindow and appending new tweets in the last time interval ‘t’ making the new time window again of ‘m’ sub-windows.

In each time window, all the tweets in that time window are utilized to detect events by the following 3 major steps:

i. **Tweet Segmentation**: Segment each tweet into non-overlapping meaningful segments (unigrams or multi-gram). The use of segments instead of just unigrams makes sure that the clusters in step (iii) contain named entities (e.g. the United States) or some semantically meaningful units (e.g. bomb blast) together instead of individual unigrams.

ii. **Bursty Segment Extraction**: Keep only those segments that are used extensively in this time window. To detect events attracting a large number of users, user frequency is also used.

iii. **Event Segment Clustering**: Cluster the bursty segments using the frequency distribution of segments and similarity of the tweets in which they were used within this time window and extract Event Clusters based on some threshold.

In above 3 phases, utilization of external knowledge base is done like Microsoft N-Gram service in Tweet Segmentation and Wikipedia in Tweet Segmentation and Event Segment Clustering to guide the event detection process. Use of user frequency is done so the event detection would be robust to the tweets of Spam and Self-Promotion.

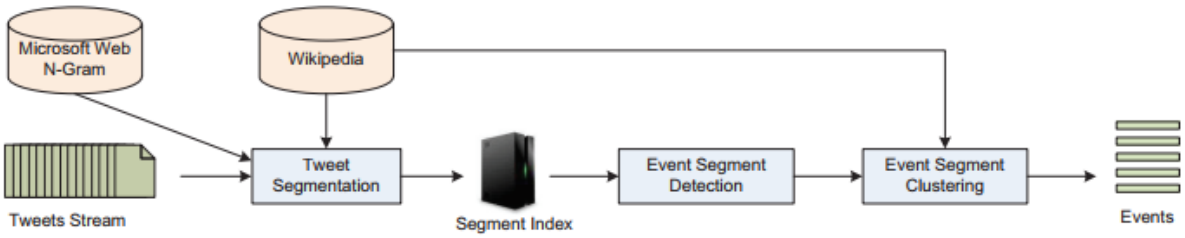

Figure 1: Segment-based Event Detection System Architecture

3.2 Tweet Segmentation

There are two approaches to tackle this task:

3.2.1. Twevent's dynamic-programming based approach in which, given a tweet 'd' ϵ T (set of all tweets), split it into 'm' overlapping segments such that each segment is either a word (unigram) or group of words (multi-gram). The objective is to maximize the sum of stickiness of segments. Mathematically,

$$\text{Argmax}_{s1,\ldots,sm}\, C(d) = \sum_{i=1}^{m} C(s_i)$$

A high stickiness of a segment means that further breaking the segment will decrease the informativeness of the segment.

The stickiness function C (·) is defined by using Symmetric Conditional Probability (SCP) for n-grams, n>=2 supported by Microsoft N-Gram service or alternately by Google N-gram viewer (2012 dataset) and Wikipedia. The Wikipedia data is also very huge but the only required data from Wikipedia is the probability of a segment being an anchor text among the pages that contain that segment. Fortunately, the processed Wikipedia data [3] is available by Prof. Aixin Sun from Nanyang Technological University (NTU), Singapore who has previously worked on a similar project of segment-based event detection.

SCP is a measure of cohesiveness of a segment 's' consisting of 'n' words i.e. s=<w1, w2,…,wn> by considering all possible binary segmentation of the segment as shown below where Pr (·) denotes the prior probability derived from Microsoft Web N-Gram service:

$$SCP(s) = \frac{\Pr(s)^2}{\frac{1}{n-1}\sum_{i=1}^{n-1} \Pr(w_1, \dots w_i)\Pr(w_{i+1}, \dots, w_n)}$$

Based on SCP, stickiness function C (·) is defined as below:

$$C(s) = L(s) \cdot e^{Q(s)} \cdot S\big(SCP(s)\big)$$

$$L(s) = \frac{|s| - 1}{|s|} \; if \; |s| > 1, otherwise \; L(s) = 1$$

L (·) is the length normalization function so that longer segments are given moderate preference. S (·) is the sigmoid function and Q(s) is the probability that 's' appears as an anchor text in the Wikipedia articles that contain 's'.

Tweet segmentation by optimizing the objective function can be done by Dynamic Programming in linear time as shown in the code snippet in Appendix A-2.

3.2.2. The problem with the above approach is that has replaced N-Gram service with the new Text Analytics API which will directly give key phrases from tweet along with its sentiment but the service allows only 5000 API calls per month. To overcome this problem, I have implemented a simple alternative segmentation method using Wikipedia page titles only. The dataset contains about 13 million page titles. While segmenting, the largest segment is selected such that it is a page name in this Wikipedia page titles list. If there is no such segment then that phrase is discarded. The advantage of this method is that it is simple to implement, faster to execute and mostly contains named entities and popular segments.

I have also used @name mentions because they mention an entity and hashtags. The @user_name is replaced by 'First_name Last_name' and considered a segment. Hashtags are split based on the capitalization or position of underscores(_) in them and the split is considered a segment. I have given hashtags three times weight compared to tweet text and named mentions because they contain a lot of information in a concise form. This more weight will help keep hashtags among bursty segments in the next part.

Moreover, I have also used a set of common phrases that appear as event clusters for most datasets. Thus filtering them out will result in a higher precision.

Examples of some tweet segmentations are given in the 4. Experimental Results section.

3.3 Bursty Segment Extraction

Once tweet segmentation is done, the next task is to find bursty segments within the current time window 't'.

Let 'N_t' denote the total number of tweets within 't', '$f_{s,t}$' denote tweet frequency of segment s in time window 't' i.e. a number of tweets containing this segment. The probability of observing segment 's' with frequency '$f_{s,t}$' can be considered as a Binomial distribution with parameters 'N_t' and 'p_s', where 'p_s' is the probability of observing segment 's' in any random time window. When 'N_t' is very large as with the case with today's tweets, this binomial distribution can be approximated to a Normal distribution with parameters $E[s|t] = N_t p_s$ and $\sigma[s|t] = \sqrt{(N_t p_s (1 - p_s))}$. Thus, a segment will be on average E[s|t] tweets. The more the tweet frequency of a segment, the more bursty it is. A segment 's' is considered bursty in time window 't' if $f_{s,t} > E[s|t]$. A segment is considered extremely bursty if $f_{s,t} > E[s|t] + 2\,\sigma[s|t]$. The bursty probability '$p_b(s,t)$' of a segment is calculated as follow:

$$P_b(s,t) = S(10\ \frac{f_{s,t} - (E[s|t] + \sigma[s|t])}{\sigma[s|t]})$$

Where S (·) is the sigmoid function. The reason for multiplying by 10 is that sigmoid smooths well in the range [-10,10].

Instead of depending entirely on tweet frequency, to incorporate user diversity, user frequency '$u_{s,t}$' is also used. I have also used two other values calculated from tweet JSON data which are '$rt_{s,t}$' and '$fc_{s,t}$'. '$rt_{s,t}$' is the retweet count of a segment 's' in time window 't' which is equal to the sum of retweet counts of all tweets containing this segment. '$fc_{s,t}$' is the follower count of segment 's' in time window 't' which is equal to the sum of followers count of all users using this segment. Instead of plainly using user frequency, the logarithm of user frequency is used because a segment used by 100 users doesn't mean it has 100 times more weight than a segment used by one user. Similarly, the logarithm of '$rt_{s,t}$' is used but I have used double logarithm for '$fc_{s,t}$' because some celebrities have hundreds of millions of followers and this should not have more effect on the bursty score than other two logarithms. Thus, the bursty weight of a segment '$w_b(s,t)$' is defined as below:

$$w_b(s,t) = P_b(s,t) * \log(1 + u_{s,t}) * \log(1 + rt_{s,t}) * \log(1 + \log(1 + fc_{s,t}))$$

Among all the segments, top K segments are selected as bursty segments based on their bursty weight. A smaller value of K would result in the very low recall of events detected, and a large value of K may bring in more noise leading to higher computational cost. Therefore, an optimal value of K is kept to be $\sqrt{N_t}$.

3.4 Event Segment Clustering

After extracting bursty segments, the next task is to cluster these segments and find events from these clusters. This process is further divided into three sub-categories:

3.4.1 Segment Similarity

For each sub-window 't_i' from time window t = <t_1,t_2,…,t_M>, the similarities of all the tweets containing the respective segments are taken into consideration for finding the similarity between two segments. Mathematically, the similarity between segments 's_a' and 's_b' can be calculated as follows:

$$sim_t(s_a, s_b) = \sum_{m=1}^{M} \mathrm{w_t}(s_a, m)\ \mathrm{w_t}(s_b, m)\ sim(T_t(s_a, m), T_t(s_b, m))$$

$$w_t(s,m) = \frac{f_t(s,m)}{\sum_{m'=1}^{M} f_t(s,m')}$$

Where, sim(T_1, T_2) is the Tf-IDF similarity between two texts T_1 and T_2. $T_t(s_a,m)$ is a concatenation of all the tweets containing segment 's_a' within sub-window 'm'. By this above formula, segments which have similar content would also be considered similar.

3.4.2 Clustering by k-Nearest Neighbor Graph

The clustering technique used here is a variation of Jarvis-Patrick [8] algorithm. In this, all segments are considered as nodes and initially, all nodes are disconnected. There is an edge between segments 's_a' and 's_b' if kNN of 's_a' contain 's_b' and kNN of 's_b' contain 's_a'. After adding all

possible edges, all the connected components are considered as candidate event clusters.

3.4.3 Candidate Event Filtering

It is observed that many event clusters detected are not actually realistic events. Thus, some filtering has to be done to eliminate these events. For this Twevent has used segment newsworthiness μ(s) and event newsworthiness μ(e) by considering external knowledge bases such as Wikipedia.

Segment newsworthiness $\mu(s) = \frac{\max}{l \epsilon s} e^{Q(l)} - 1$ where 'l' is any sub-phrase of 's', and Q(·) is the prior probability that 'l' appears as anchor text in Wikipedia articles that contain 'l'.

Event newsworthiness $\mu(e) = \frac{\sum_{s\epsilon e_S} \mu(s)}{|e_s|} * \frac{\sum_{g\epsilon E_e} sim(g)}{|e_s|}$ where 'E_e' is a set of edges that are retained after applying clustering, and sim(g) is the similarity of edge 'g'.

After calculating event newsworthiness, all clusters which have newsworthiness more than (1/τ) of max newsworthiness are considered as events and then their summary is calculated based on the text of tweets containing these segments using a python module. I have obtained optimal results by keeping τ=4.

4. Experimental Results

The data on which the whole event detection is done is from October 11, 2012, to October 22, 2012. Each hour had roughly 200,000 English tweets resulting in about 2.5 million tweets per day. In today's date, this number would be even larger. The Wikipedia file contains page titles as of March 2018. This file is used in the tweet segmentation process. Following are some examples of tweet segmentation with triple weight to hashtags:

1. 'VP Joe Biden and Paul Ryan will be seated at the debate tonight #VPDebate' → 'joe biden', 'paul ryan', 'seated', 'debate', 'tonight', 'vp debate', 'vp debate', 'vp debate'
2. 'LIVE NOW The Vice Presidential Debate Paul Ryan vs Joe Biden #VPDebate #ClearEyesFullHeartsCantLose' → 'live', 'vice presidential debate', 'paul ryan', 'joe biden', 'vp debate', 'vp debate', 'vp debate' ,'clear eyes full hearts cant lose', 'clear eyes full hearts cant lose', 'clear eyes full hearts cant lose'
3. 'Watching the vice president debate with @BryanSelleck #gopaulryan' → 'watching', 'vice president', 'debate', 'bryan selleck', 'gopaulryan', 'gopaulryan', 'gopaulryan'

In this project, the time window was kept of 1 day consisting of 12 sub-windows each of two hours. For finding the expected probability of any segment being in any random window (p_s), the same dataset of October 11, 2012, to October 22, 2012, was used and all tweets were segmented with the above-mentioned segmentation method. Then segment frequency of each segment was counted and divided by a total number of segments to find segment probability for all segments.

Using this above segmentation technique, top $k = \sqrt{N_t} = 1{,}328$ bursty segments were extracted where $N_t = 1{,}765{,}856$ is total number tweets containing 470,249 unique segments within the time window containing tweets from 12 October 2012. The top 10 bursty segments were as under along with their bursty score:

1 vp debate 17.7020

2 joe biden 15.3149

3 vpdebate 15.1605

4 reasons why we dont get along 14.7204

5 v pdebate 14.4745

6 paul ryan 14.2069

7 debates 14.0192

8 wan na 13.7770

9 debate 13.7561

10 dont know 13.2206

The segments marked bold actually represent an event which happened on October 12, 2012. The event was a Vice Presidential debate between Joe Biden and Paul Ryan.

After finding these 1.328 bursty segments, they were clustered based on similarity measure mentioned in section 3.4.1. Following are the detected with k = 4 neighbors while clustering. Notice that the last example on next page is not an event but still recognized after clustering.

- **Vice Presidential debate between Joe Biden and Paul Ryan** → ['vp debate', 'joe biden', 'debates', 'debate', 'biden', 'vice presidential debate', 'obama', 'barack obama', 'vp biden', 'msnbc', 'vp debates', 'cnn', 'team joe', 'facts matter', 'obama biden', 'last resort', 'get glue', 'details matter', 'batb', 'cnn debate']
- **About Nobel peace Prize and about EU and wining the same prize** → ['nobel peace prize', 'nobel', 'european union', 'bahrain', 'human rights', 'peace prize', 'nobel prize']
- **Justin Bieber and Nicki Minaj's new song Beauty and a beat music video released on YouTube** → ['justin bieber', 'beauty beat', 'nicki minaj', 'beauty and a beat video', 'baab', 'noon', 'beaut yanda beat', 'beautyandabeat', 'beauty and a beat']
- **Pittsburgh Steelers vs. Tennessee Titans TNF football game** → ['martha raddatz', 'chris brown', 'steelers', 'nfl', 'titans', 'jim lehrer', 'chris matthews', 'steeler nation', 'chris johnson', 'gary johnson and jim gray', 'tnf']
- **New York Yankees losing a playoff baseball game against Orioles in American League Division Series (ALDS)** → ['alds', 'yankees', 'tigers', 'would like', 'orioles', 'pro life', 'buc kle up', 'dont love', 'postseason', 'would love', 'new york', 'much love', 'say love', 'athletics', 'whole world', 'new york yankees', 'want life', 'justin verlander', 'still love', 'love much']

- ['would say', 'people say', 'dont say', 'never seen', 'say something', 'whatever say', 'say say', 'said never', 'never say anything', 'say anything']

Comparison of metrics using Twevent's and my approach is given below for tweets from 11th October, 2012 to 17th October, 2012:

Method	No. of events detected	Precision	Duplicate Event Rate
My Approach	78	87.25%	14.10%
Twevent	42	80.32%	16.67%

Precsion is defined as the fraction of the detected events that are related to a realistic event. **Duplicate Event Rate (DERate)** is the percentage of events that have been duplicately detected among all realistic events detected. I am not able to calculate **Recall** because although the dataset (Events 2012 [3]) that I used had mentioned the events their model detected, there were a lot of events (approximately 50 new events within the period of 1 week) that were detected by my model, and many of their detected events were not detected by my model of the Twevent model. This makes me believe that their results are not an exhaustive set of all the events that happened during the interval of tweets.

The total time taken to find events in a time window of 24 hours on a laptop with Intel Core i5 – 4200U CPU @ 1.6 GHz 2.3GHz and 8GB RAM was 4.5 hours. A majority amount of time was consumed in finding similarities between every pair of segments. This can be improved by using multithreading.

5. Conclusion

Twitter has experienced an explosive increase in both users and the volume of information in the recent time. This has attracted great interests from both industry and academia. Many private and/or public organizations have been reported to monitor Twitter stream to collect and understand users' opinions about the organizations, recent events, and popular discussions.

In this project, I've tried to detect broad categories of popular events. The results have been promising so far. There is scope for improvement in terms of more pre-processing and training over the bulk of data. This shall be done in near future.

References

1. Li, C., Sun, A., Datta, A.: Twevent: segment-based event detection from tweets. Proceedings of the 21st ACM International Conference on Information and Knowledge Management (2012). http://dl.acm.org/citation.cfm?id=2396785
2. TwiNER: Named entity recognition in targeted twitter stream: https://www.researchgate.net/publication/254464404_TwiNER_Named_entity_recognition_in_targeted_twitter_stream
3. Andrew J. McMinn , Yashar Moshfeghi , Joemon M. Jose, Building a large-scale corpus for evaluating event detection on twitter, Proceedings of the 22nd ACM international conference on Conference on information & knowledge management, October 27-November 01, 2013, San Francisco, California, USA
4. Wikipedia Keyphrases dataset: https://www.ntu.edu.sg/home/axsun/datasets.html
5. Twitter dataset: https://archive.org/details/archiveteam-twitter-stream-2017-09
6. Google Books N-Gram Viewer: http://storage.googleapis.com/books/ngrams/books/datasetsv2.html
7. Phrasefinder API to search Google N-Gram dataset: http://phrasefinder.io
8. pyTweetCleaner: https://github.com/kevalmorabia97/pyTweetCleaner
9. R. A. Jarvis and E. A. Patrick. Clustering using a similarity measure based on shared near neighbors. IEEE Trans. Comput., 22(11):1025–1034, Nov. 1973.
10. https://stackoverflow.com

Appendix A

1. Sample json tweet before and after preprocessing

Raw Data:


```
{
"created_at":"Sun Sep 10 06:30:00 +0000 2017",
"id":906766690821668864,
"id_str":"906766690821668864",
"text":"I played the Sandy Caps mini game in Paradise Island 2, and my score was: 317
#GameInsight #ParadiseIsland2",
"source":"\u003ca href=\"http:\/\/www.game-insight.com\/\"
rel=\"nofollow\"\u003eParadise Island 2\u003c\/a\u003e",
"truncated":false,
"in_reply_to_status_id":null,"in_reply_to_status_id_str":null,"in_reply_to_user_id":null,"in_
reply_to_user_id_str":null,"in_reply_to_screen_name":null,
"user":{
        "id":906130974655766529,"id_str":"906130974655766529","name":"Yulia Astuti",
        "screen_name":"YuliaAs777", "location":"Yogyakarta, Indonesia", "url":null,
        "description":"Simple", "translator_type":"none", "protected":false, verified":false,
        "followers_count":3, "friends_count":51, "listed_count":0, "favourites_count":0,
        "statuses_count":97,"created_at":"Fri Sep 08 12:23:54 +0000 2017",
        "utc_offset":null,"time_zone":null,"geo_enabled":false,"lang":"en","contributors_e
        nabled":false,"is_translator":false,"profile_background_color":"F5F8FA","profile_b
        ackground_image_url":"","profile_background_image_url_https":"","profile_back
        ground_tile":false,"profile_link_color":"1DA1F2","profile_sidebar_border_color":"C
        0DEED","profile_sidebar_fill_color":"DDEEF6","profile_text_color":"333333","profil
        e_use_background_image":true,"profile_image_url":"http:\/\/pbs.twimg.com\/pr
        ofile_images\/906134056353513472\/ORRZ0AXG_normal.jpg","profile_image_url
        _https":"https:\/\/pbs.twimg.com\/profile_images\/906134056353513472\/ORRZ
        0AXG_normal.jpg","profile_banner_url":"https:\/\/pbs.twimg.com\/profile_banner
        s\/906130974655766529\/1504874167","default_profile":true,"default_profile_ima
        ge":false,"following":null,"follow_request_sent":null,"notifications":null
},"geo":null,"coordinates":null,"place":null,"contributors":null,"is_quote_status":false,"qu
ote_count":0,"reply_count":0,"retweet_count":0,"favorite_count":0,
"entities":{
        "hashtags":[{"text":"GameInsight","indices":[78,90]},{"text":"ParadiseIsland2","indi
        ces":[91,107]}],"urls":[],"user_mentions":[],"symbols":[]
},"favorited":false,"retweeted":false,"filter_level":"low","lang":"en","timestamp_ms":"1505
025000659"
}
```

Preprocessed Data:


```
{
"created_at": "Sun Sep 10 06:30:00 +0000 2017",
"text": "played sandy caps mini game paradise island score",
"user": {
          "id": 906130974655766529,
          "followers_count": 3
          },
"retweet_count":0,
"entities": {
          "hashtags": [“GameInsight","ParadiseIsland2"],
          "user_mentions":[]
          },
}
```


2. Dynamic Programming approach by Twevent for Tweet Segmentation

```
def tweet_segmentation(self, tweet, max_segment_len=3, e = 5):
    words = self.tokenize(tweet)
    n = len(words)
    S = [[] for i in range(0,n)] ## S[i] = top e possible segmentations of first 'i' words
    for i in range(0, n):
        if i < max_segment_len:
            S[i].append( ([words[0:i+1]], self.get_stickiness(words[0:i+1])) )
        j = i
        while j>=1 and i-j+1<=max_segment_len:
            t2 = words[j:i+1]
            for segment in S[j-1]:
                new_seg = []
                for s in segment[0]:
                    new_seg.append(s)
            new_seg.append(t2)
            S[i].append((new_seg, self.get_stickiness(t2) + segment[1]))
        S[i] = sorted(S[i], key = lambda x: x[1], reverse=True)[0:e]
        j -= 1
    return S[n-1][0][0]
```