Smart Segmentation

Sameet Sapra
University of Illinois at Urbana-Champaign
Urbana, IL
ssapra2@illinois.edu

Andrew Doherty
University of Illinois at Urbana-Champaign
Champaign, IL
adohert2@illinois.edu

Richard Weeks
University of Illinois at Urbana-Champaign
Spokane, WA
rtweeks2@illinois.edu

ABSTRACT

In this paper, we introduce an end-to-end solution for a Chrome extension that segments a text into phrases of interest around a certain word. We denote these phrases of interest as "smart segmentation." This paper will discuss how we generate, rank, and display smart segmentations. We will compare our approach with existing solutions. We also look at the performance of our moldel and UI and discuss how smart segmentation can inspire future work in extracting structured data from the web.

1. INTRODUCTION

In this section, we will introduce what smart segmentation is and why smart segmentation is both an important and difficult concept.

1.1 What is Smart Segmentation?

We define the problem of smart segmentation as finding the boundaries between phrases within a text that are most important to a user. Segmentation itself is not a new concept to the browser. In fact, every browser already segments text. When a user double-clicks any word on their browser, the browser knows that the user is interested in the word, so it highlights the word. If a user decides to click one more time, the browser then highlights the entire block of text. Most browsers also provide actionable functionality regarding this selection, such as the ability to search among different search engines. However, this segmentation is very basic because it only differentiates the boundary between a single word and the entire text block. Smart segmentation aims to bridge that gap by suggesting boundaries that reflect the context surrounding the word.

1.2 Why is Smart Segmentation important?

Segmentation on the Web is both important and difficult because the Web is not designed for convenient, automated consumption of information. There is little reason for product vendors to make information on web pages easily digestible for third-party software. The ease of extracting information is significantly less important than the visual appeal of the page, because online vendors are rewarded, often with additional sales, for making their web pages visually appealing to potential customers. In fact, vendors might be penalized by comparison shopping, diminished advertising revenue, and/or reduced up-sale/cross-sale opportunities if they make the product data on their pages easy to extract with non-browser software.

Big data continues to get bigger as the Internet grows in size, with billions of users and even more web pages. The data that exists on the web, though, is primarily unstructured data, because the most common form of data on the web is raw text. Raw text is inherently ambiguous, which makes it difficult to store in a more efficient, queryable form. This makes it more difficult to work with, thus leaving most data explorers to look for data in structured data.

If we could divide this raw text into identifying entities and attributes, we could allow for more analysis to be conducted on this unstructured data, such as data extraction or clustering.

1.3 Applications of Smart Segmentation

A very simple application of smart segmentation is *intelligent* highlighting of important terms on web pages. A user shopping for headphones on various websites may be reading reviews of the headphones. Instead of highlighting the entire product, copying the segment, and opening a new browser window to search for the item, a smart segmentation tool could have already recognized the headphones and provided an option for the user to simply click the link.

Smart segmentation can also benefit web crawlers. As the web crawler encounters and processes web pages, it needs to extract relevant information from the HTML of the page for later querying. This extraction is a tedious, error-prone process because web pages tend to be structured for presentation rather than data transfer and the presentational form is completely determined by the author of the page. To deal with this, the crawler needs tools to identify the relevant attributes of the page; smart segmentation can be an important aspect of this process.

This could also be useful for data analytics as it will allow a more fine grained comparison of products. As an example, it is much easier to compare 32GB to 64GB than to compare "Google Pixel XL 32 GB Really Blue unlocked" and "Samsung Galaxy S8 64 GB Midnight Black Boost Mobile". By having certain data values separated, it will become eas-

ier to analyze and compare products by their attributes in order to give users a better comparison that is meaningful. Data analysts will be able to focus on understanding the data rather than ensuring that the given data is accurate and of the same form.

Allowing web browsers to understand data on web pages would help in highlighting useful information, filtering and sorting data, and exporting data into a tabular structure like Excel. It would allow developers to more intelligently parse the vast amounts of unstructured data, and expand on reliable uses of data sources that already exist on the web instead of solely relying on structured data.

1.4 Why is Smart Segmentation Difficult?

The biggest challenge in engineering solutions for smart segmentation is that web page publishers do not benefit from publishing the data presented by a web page in computable form. Web pages are designed for presentation to a user, so publishers typically structure their data to maximize impact with their customer base. This makes it difficult to identify the boundaries between relevant and irrelevant text on the page and even moreso to have a single smart segmentation model that will work on all websites. For the purposes of this research experiment, we chose to narrow the domain of the websites to e-commerce shopping websites as a more manageable problem scope. This may come to affect our design of the smart segmentation model. In future iterations, we may widen our scope and re-visit the fundamental model assumptions.

Finally, data segmentation depends on multiple cues, including DOM analysis, natural language processing, visual analysis, and the user interface. Creating a good solution that solves data segmentation requires significant advances on all of these engineering challenges.

2. EXISTING FIELD WORK

In this section, we take a survey of the current state-of-the-art techniques in text entity recognition and existing solutions to better understand different tactics to tackle smart segmentation.

2.1 Stanford Named Entity Recognizer

Stanford's NLP group has a named entity recognizer that can classify words in a given text to entities.[5] Stanford did an experimental study for named entity recognition in tweets and used POS (part of speech) tagging. The first lesson they learned about recognizing entities is that sufficient context around words is needed: while a dataset like a news source may have related words and more context on a page, a singular, independent record like a tweet is much more difficult. Stanford explored POS tagging as well as shallow parsing, the task of segmenting noun phrases, verb phrases, and prepositional phrases. The paper's authors discovered the challenge of lexicographical variations of words and found that led to worse results on mal-formatted text like tweets compared to text pages with proper grammar, like online news articles. The POS Tagger, Chunker Named Entity Recognizer is thus limited to only classify three entities: person, organization, and location. A tool like this would only be able to identify nouns or adjectives, which are potential attributes, and at best, shallow parsing is still limited to noun phrases or verb phrases. In dealing with e-commerce websites, shallow parsing may have trouble distinguishing between proper nouns and noun phrases, which could lead to accuracy problems. This solution is not fine-grained enough to identify specific entities and group similar attributes across different text inputs.

2.2 NoDoSE

Brad Adelberg's NoDoSE tool provides a GUI tool for decomposing a text-based document into multiple tables. [1] The tool is semi-automatic, as it requires the user to outline interesting regions of the text page and describe the semantics of the webpage. Then, the input is converted into a data spreadsheet and can be input into some sort of data analysis program or a database. We feel that this type of user input depends too much on the user and makes the process of identifying segmentations less intuitive for the user. Adelberg's solution also bears remarkable resemblance to a data view for the web because the contents of modern web pages has become more diverse even within the same page due to aggregation of data from many backend systems.

2.3 Conditional Random Fields

Lafferty, McCallum, and Pereira from the University of Pennsylvania uses purely statistics over modelsto label entities in natural language processing.[3] The paper explains how *label bias* can negatively affect the accuracy of Markov models, and shows how CRF's single model for joint probabilities of entire label sequences avoids this label bias. This approach contains a lot of statistical background that has driven our approach for smart segmentation. The addition of global observations like page topic keywords determined by TF-IDF, could serve to improve upon this label bias as well.

2.4 Source Data: The DOM Tree

Because it is the only available source of the information for a Chrome extension, our solution must draw some of the input from the web page's DOM tree, which is exposed to the code of our Chrome extension. The DOM tree is structured for the task of presenting data to the user in the visual format selected by the publisher. As mentioned previoiusly, this presents a challenge because the data of interest to the user is not clearly identified for consumption by our Chrome extension. For a specific example, while Amazon.com and Best Buy both offer online retail of an overlapping set of products and display search results and products in a way that is visually similar, the structure of the DOM trees on the two sites is, in many ways, dissimilar.

An additional challenge faced in smart segmentation is that web page publishers can change the format of their pages at any time and in any way that still produces a page that conforms to HTML. Thus, even for a given domain on the Web, we cannot rely on heuristics that happen to successfully identify our target data at one given time. For example, one Web site could wrap the description of a product in a `<div>` with the CSS class *description*. Using or allowing heuristics based on this fact would have two negative consequeces: first that such rules have to be specific to and discovered for each Web site, and second that our solution could break as soon as any site making use of such rules was reformatted with a different presentation format. To produce the most lasting, useful results, we chose to focus

on heuristics that do not depend on arbitrary choices of the publisher but rather on the visual experience of the users.

3. SOLUTION

Our overall approach to solving smart segmentation is a type of unsupervised learning where we analyze frequent phrases that occur on the page to curate a ranked list of suggested phrases. When a user indicates a word to segment, the text on the page, the user-selected word, and the context surrounding the word are passed as input to a model. The model needs to ingest this data and provide relevant segmentations related to the context surrounding the word. Several concepts emerged during our research of existing tools and research: sequence mining, pattern mining, phrase mining, and TF-IDF. We'll discuss the conclusions of each of these concepts as separate sections.

3.1 Pattern Mining

We evaluated the `pymining` Python library as an implementation of frequent pattern mining for our project. Our results showed that it was too slow as it compared all combinations of words against each other, resulting in an exponential time algorithm. We looked into frequent pattern mining and frequent phrase mining, implemented in many Python libraries like `nltk`, `prefixspan`, and `gensim`. Pattern mining is tasked with finding statistically significant patterns between different data examples based on how the values occur in a sequence.[4] In fact, frequent pattern mining looks at TF-IDF as one of the primary sources to train and evaluate a document. Through our initial model comparisons, we ended up using the Python library `gensim` to identify phrases using `gensim`'s Phrases API, which uses frequent pattern mining under the hood.

3.2 Ranking Function

Once the segmentations are generated, the segmentations are passed to the ranking function. The ranking function's role is to determine the order in which segmentations should be displayed to the user. We always want the first segmentation to be the singular word that the user right-clicked. This is because the segmentations are all centered around that word, so the smallest segmentation to start with is the word that the user right-clicked. Similarly, the last segmentation should be the entire record from which the user's right-clicked word was derived from. Otherwise, for all other segmentations, the current ranking function uses primarily two metrics:

1. number of words in a segmentation
2. number of characters in a segmentation.

Our ranking function focuses on the number of words in a segmentation as a segmentation with more words is generally less common - so if this phrase has a high frequency, then statistically, the phrase corresponds to a more meaningful segmentation.[1] The reason for including the number of characters is due to the reasoning that longer words are typically less common than shorter words, so a phrase with longer words has a higher chance of being meaningful than a phrase with numerous short words. In our current implementation, we sort the segmentations in increasing length so that segmentations always *expand* in increasing length. We'll see why this is the case again when we discuss the user interface.

3.3 Word Shape

An interesting problem that we tackled during designing our ranking function and generating segmentations is the notion of *word shape*. Phrases such as "Intel Core i3-4510U" and "Intel Core i7-8850X"[2] are quite similar to the user, yet to a computer, they cannot be recognized as the same because the numbers make the phrases unique. Instead, our approach was to analyze the word shape of text and clean it so that the model recognizes these phrases as the same: "Intel Core i####-####U". The benefit of this is that frequent pattern mining treates these tokens as the same word, so that the numbers do not negatively affect the TF-IDF counts used in frequent pattern mining.

The difficulty associated with this problem was re-constructing the original phrase from these *cleaned* results, as this algorithm alters the phrases so that they don't correlate to the same phrases. The model seemed to improve and fit segmentations better compared to without this approach, so this problem was a good improvement to make to clean the input data before segmentations were generated.

3.4 Fetching and Displaying Ranked Segmentations

Our initial UI had the user selecting text as the first step to getting segmentations. Based on feedback from our mentor on this UI, we decided it was not desirable to require the user to manually specify the boundaries of our segment search within the text of the page by highlighting it, primarily because certain text (e.g. links) in the browser can be hard to highlight and text selection can be more awkward for the user. Instead, we changed our approach to deduce a node in the DOM tree containing the target text using some heuristics intrinsic to web page structure, then extract its text content as the full record context for our segmentation model. This provides us cleaner data that is focused on a single item rather than the entire list of items displayed by many pages.

One signal we were able to leverage for our heuristics relates to the proportion of the page's space consumed by various elements in the page. By combining the consumed page area and the CSS class names applied to a candidate element and other elements on the page, we were able locate a boundary around a target point on the page within which all data referenced the same item. This was important on a multi-item search results page for two reasons. First, users would not typically want segments involving two or more search results. Second, it was important for optimal performance of our segmentation algorithm to minimize the size of the input data.

Another source of useful text for analysis and hints about the meaning of DOM substrees are *Accessible Rich Internet Applications* (ARIA) annotations attached to HTML elements of the page. We leveraged these indicators to get text

[1]This assumption can be made because of the domain we assumed this tool would be used in - e-commerce websites. In a product listing record, typically every word is important, and the longer the phrase, the less common that phrase is. This does not hold true for all pages however. In future iterations of this model, this assumption should be re-visited.

[2]These processors are made up numbers. They do not correlate to real processors

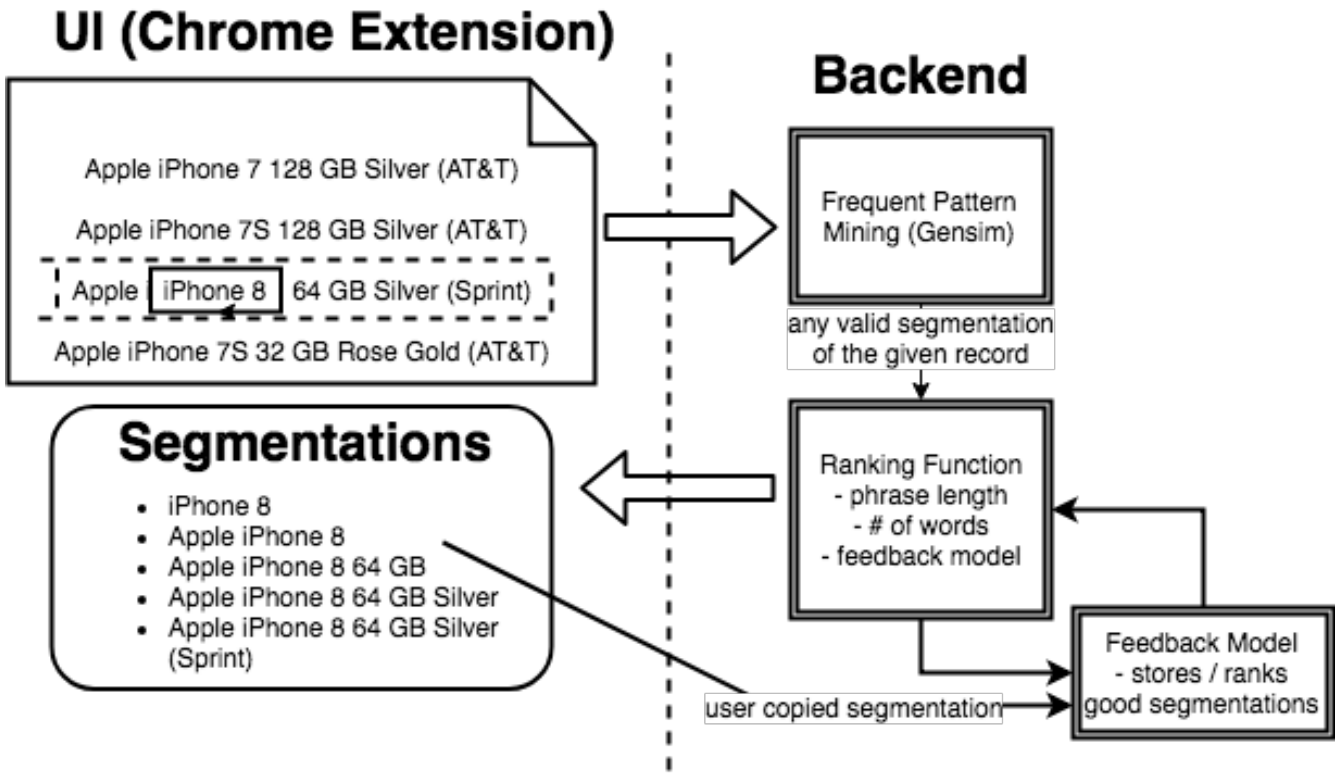

Figure 1: Solution Architecture

from the page where the page publisher was using DOM tricks to achieve graphical displays; this was particularly used with *star ratings* for products. An additional *ARIA* attibute allows our solution to ignore portions of the page included only for visual effects. We believe that these *ARIA* attributes are a useful, stable source of good information for segmentation because web page publishers who provide these attributes want their pages to be accessible to users with visual disabilities. However, this introduces danger that web page publishers will discover tricks with *ARIA* to either produce a specific, non-visual effect or to obfuscate the presented data.

Use of these heuristics allows our solution to identify an appropriate context in which to apply segmentations with a level of accuracy that does not impact user experience. It does so without comparative analysis of multiple pages from the same site – as might occur in a rudimentary Web crawler – which might not be easily available in our target use case. By doing so it avoids the dependencies on unreliable assumptions and *a priori* knowledge about the site chosen by the user.

Our solution also manipulates the DOM tree to better identify the user's indicated starting point. When a user right-clicks a hyperlink, the Chrome browser highlights the entire link, which is different behavior than for other text on the page. To allow our solution to be consistent regardless of the presence of a hyperlink, our extension wraps each word within a link in a . Our code can then determine which of these elements is the true target of the modified right-click event. This solution is not perfect, however, and can sometimes alter the way pages display links.

To show the result of our segmentation algorithm, our extension takes the identified set of substrings from the context and re-locates the string within the DOM tree to the best of its ability. The user can switch between the identified segments with the arrow keys. This operation is complicated by several factors pertaining to the DOM tree and our segmentation process:

- text can be spread out across multiple nodes
- our context collection algorithm makes certain modifications – such as whitespace normalization – to the collected text; segment recognition may also make changes – such as non-ASCII character suppression – for which the search code must account
- the identified segments are not necessarily unique within the identified context
- the identified segments do not necessarily share boundaries with elements.

We use a modified implementation of the Knuth-Morris-Pratt string search algorithm to find the target substring while only walking forward through the DOM tree structure. The algorithm is used to find occurrences of one string within another string in an efficient way.[2]

3.5 User Interface

Our ranking function imposes a strictly increasing order of segmentations so that the best segmentations are returned

at each available length. The reason this was to make it easy for the user to navigate through segmentations, which we will discuss here.

On the chrome extension, the user has six simple actions they can do at any given moment once they have entered segmentation mode:

1. $\Uparrow$: Cycles to the previous best segmentation of decreasing length
2. $\Downarrow$: Cycles to the next best segmentation of increasing length
3. $\Leftarrow$: Cycles to the previous best segmentation of the same length
4. $\Rightarrow$: Cycles to the next best segmentation of the same length
5. *Space*: Copies the current segmentation to the clipboard
6. *Esc*: Escapes segmentation mode

Notice that all actions in segmentation mode make use of the keyboard. This is intentional to provide an easy, non-intrusive way to sift through all segmentations while remaining an intuitive part of the browser. This was one of the goals of smart segmentation: to provide a complementary way of viewing segmentations of a word.

4. EVALUATION

Evaluation of our smart segmentation model is done by running our model against various e-commerce website phrases with training data from Best Buy and Amazon pages on iPhones and ASUS laptops.

4.1 Results

The model successfully identifies bigrams of words and also recognizes and groups associated words together based on the frequency of those patterns. Our ranking function filters out segmentations that do not include the user's selected word from the UI. As discussed earlier, initial heuristics on the ranking function favored segmentations with fewer words yet a greater number of characters. Our intuition for this was based on the notion that a long phrase with few words will likely be an important entity. In this section, we look at how ranked segmentations can be evaluated so that there is numerical value associated with the segmentations.

4.2 Similarity with Search Engine

While the smart segmentation model recommends what the model believes to be good segmentations based on various heuristics discussed, the model is following an unsupervised learning model, where there is no typical evaluation of the accuracy of the structure output by the model. Instead, another entity - most commonly a user - must look at the results and identify if a segmentation phrase is good or not. This is similar to evaluation of documents within a search engine, as results returned by a search engine can be relevant or non-relevant to a user based on a search query, and the results should be viewed in decreasing order of importance to the user.

4.2.1 Precision and Recall

The first two metrics we should be concerned with is Precision and Recall. Precision is the fraction of relevant segmentations among all segmentations generated. Recall is the fraction of relevant segmentations among all *relevant* segmentations. These are common metrics used in unsupervised learning problems with ranking of results. This is because what is deemed relevant is ultimately determined by the user and no one else.

4.2.2 Accuracy

The percentage of segmentations the model generates that are part of the overall segmentation results is also important. This is simply referred to as the *accuracy* of the model. Unlike search engine document ranking, this metric is important to calculate as the model has to first identify segmentations in the phrase before ranking them. Using just precision and recall captures the ranking of segmentations returned, but the actual segmentations of the model are important to measure alone, regardless of rankings. Our model achieves around a 65% accuracy on a pre-determined test set of e-commerce websites and records.

5. TIME

Our model generates segmentations for a given phrase as fast as 200 milliseconds for a single webpage worth of training data and a record for a product entry of 20 words. That number can increase to a second for an average user's sessions worth of page data. Any page data stored beyond that does not contribute to overall improvement of segmentation results.

6. LESSONS LEARNED

In this section we discuss overall lessons learned from building a Chrome extension and making a smart segmentation model.

6.1 Data Storage

To experiment with different levels of segmentation data, we learned that we could run three iterations of the model on local data (text that existed solely from the current page), session data (text from the session of pages that a user has visited), and global data (text from all pages ever visited). This global data was stored on our backend server. Throughout the project, we determined that the global model will occasionally outperform the local model, but on average the models perform the same or very close to the same. However, we learned that the global data model would always be increasing in size, which would slow down our model. Additionally, we did not see a huge benefit in segmentation phrases being significantly more relevant to the end user. There is also an obvious data privacy issue if page data is sent to a remote server every time segmentations are generated. Since the performance gain is minimal at best and the time it takes to compute results increases significantly, the local model tends to be the better choice, especially because with a tool like this, the time it takes to compute and return results is important as the user does not want to wait for results to be returned.

Thus, for simplicity, speed, and data privacy, no training data should be stored on disk as it only increases the time

and further training data does not greatly improve segmentation results. Instead, several pages worth of training data can be stored in memory and passed to the model to train.

6.2 Intuitive User Interface

We also learned that a strong, intuitive, clean, and non-intrusive user interface is of utmost importance in sifting through segmentations. The user interface went through several iterations. In iteration 1, we believed that the user should be able to highlight any text on the page, and the segmentations should be generated via a modal that blocked all user-interaction and allowed the users to view the segmentation, the phrase, and an action to copy to the clipboard. However, this proved problematic as a link cannot be arbitrarily highlighted without pressing another key on the keyboard. Thus, it made highlighting words difficult for the user. In iteration 2, we wanted to make use of the context-menu (the menu that appears when you right-click on a webpage) and asynchronously fetch and store the segmentations there so that the user could see them as soon from a right-click on the page. However, we felt that this was too hidden for the user and it could lead to long lists of segmentations without any custom CSS styling to make it more intuitive.

Thus, finally we decided upon using right-clicking and asking for segmentations by clicking an option on the context menu. We also thought about implementing a loading screen while the segmentations were fetched, as the browser asynchronously fetches and displays the segmentation mode with the ranked segmentations; however, it also led to an obtrusive user experience.

7. FUTURE WORK

In this section, we discuss remaining work that can be done to improve upon the initial solution of smart segmentation, and preview what's to come in future iterations of smart segmentation.

7.1 Transpiling Python

Our current implementation of smart segmentation requires a backend Python server to run `gensim`. If this model were transpiled to JavaScript, the entire backend server would not be needed, as the model can fit into the Chrome extension. Segmentations could be generated without the added network latency of communicating with an external server. This would also alleviate issues of data privacy as the page's text is not sent anywhere else - it is only read into the Chrome extension itself.

7.2 Model Feedback

The current model returns numerous valid segmentations currently, yet some of these segmentations may not be the most pertinent segmentations, or they may just not be what the user was looking for. In order to improve this, the next step in this project is to add a user feedback model. When a user copies a segmentation to their clipboard, the segmentation will be considered good since the user seems to like what was selected and plans to use it elsewhere. However, if the user scrolls past a segmentation, that segmentation will be considered bad. The decision of a good or bad segmentation will then be sent to a machine learning model which will use this feedback to better select future segmentations for the user. This data can also be aggregated among users and used to determine which segmentations are generally good and which ones are generally bad, and then this will help determine which segmentations the model will score higher.

This can be introduced into our current design by having an associated weight vector for each phrase, and assigning a value between -1 and 1 to indicate the overall preference of that segmentation by all users collectively. The feedback model needs to be collected globally for all users of the chrome extension, so the backend server would still be needed to fetch and store overall segmentations of value. Upon segmentations, the chrome extension should be able to (optionally) retrieve the associated weight vector from a backend server and apply the weights to the segmentations so that user feedback can be applied to the

7.3 Frequency Counts

In order to improve the model and ranking of segmentations, the model should use the frequency counts of the phrases. The current model uses number of words and number of characters in ranking segmentations, however using the frequency counts will help give more accurate results since a phrase with more counts should be ranked better than a phrase with the same number of words and characters but a lower frequency. The number of occurences of a segmentation will not only be a tie-breaker for segmentations, but should also contribute more than the current metric of number of characters. A more common segmentation is most likely more important to the user than a segmentation with a longer word that hardly appears.

8. CONCLUSION

In this paper we have defined the problem of smart segmentation. We have looked at why it is difficult and discussed various approaches in related work around smart segmentation. We discussed pattern mining as an initial solution for smart segmentation, and we discussed possible heuristics to rank segmentations, such as frequency counting, overall segmentation length, and number of words in a segmentation. We discussed how the feedback model can fit in this model and proposed a user interface in the form of a Chrome extension that can segment words based on Smart segmentation inspires further work and research into WebDataView. WebDataView is a way of building a data table from unstructured data on the web by writing a query. This data can be exported into a csv spreadsheet for data ingestion. Our smart segmentation tool can be found here: https://github.com/sameetandpotatoes/smart-segmentation

9. REFERENCES

[1] B. Adelberg. Nodosea tool for semi-automatically extracting structured and semistructured data from text documents. In *ACM Sigmod Record*, volume 27, pages 283–294. ACM, 1998.

[2] D. E. Knuth, J. H. Morris, Jr, and V. R. Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977.

[3] J. Lafferty, A. McCallum, and F. C. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[4] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach.

IEEE Transactions on knowledge and data engineering, 16(11):1424–1440, 2004.

[5] A. Ritter, S. Clark, O. Etzioni, et al. Named entity recognition in tweets: an experimental study. In *Proceedings of the conference on empirical methods in natural language processing*, pages 1524–1534. Association for Computational Linguistics, 2011.