Learning Character Graphs from Literature

Sumit Gogia, Min Zhang, Tommy Zhang

December 14, 2015

Abstract

We present a method for extracting salient characters and recognizing salient character relationships from literature. As opposed to previous work which extracts *social* networks by examining dialogue *given* characters, our trained system finds characters and relationships directly from raw novel text. We take a novel supervised learning approach where we retrieve labels from a simple, digestible online source (Sparknotes) to train classifiers that identify characters as salient or pairs of characters as related. Initial results show that even basic classification methods produce good performance for salient character extraction, while relationship detection warrants significant improvement[1].

1 Introduction

Literary scholars, when comparing different works of literature, frequently examine major characters and their relationships. These examinations cover attributes such as number of major characters, names of characters, and types of relationships in each novel to make higher-level statements about novel form.

Unfortunately, at current, the approaches to such examination are typically manual and require close reading, preventing comparison of the large existing bodies of literature. While statements can and have been made for a small manageable subset, this notably results in a heavily-studied canon and a largely-ignored general body of literature [15]. A conclusion literary scholars have come to is this: we need efficient *automated* methods for extracting higher-level representations from literature [14].

Recent years have seen work founded in this realization, both for character analysis as well as other high-level analyses such as plot extraction. In character analysis, methods have focused on *social network extraction*, as many literary theories involve the communication between characters [3, 13, 17]. These methods take lists of characters in a novel and then describe their relationships in a graph, with edges connecting characters indicating relationships. They utilize *dialogue* as the basis for their analysis, extracting dialogue from raw text

[1] All code for this project can be found at `https://github.mit.edu/summit/chara-extractor`

and attributing it to characters to compute edge weights and node sizes.

However, despite the success of these methods, they do not address many of the needs for automated character examination, including automated extraction of many attributes noted above. The goal in this paper is to supplement automated social network extraction with automated methods for extracting two useful representations:

1. A list of major characters in a novel.
2. A list of pairs of major characters with a salient relationship in a novel.

Importantly, these representations are different from those found in social network extraction. Firstly, social network extractors have character lists as input instead of output - our character extractor can feed output into social network extractors. Secondly, social network relationships are found through dialogue only as opposed to the text entirety. A graphic describing the representations found by social network extractors and our extractors is given below for clarity.

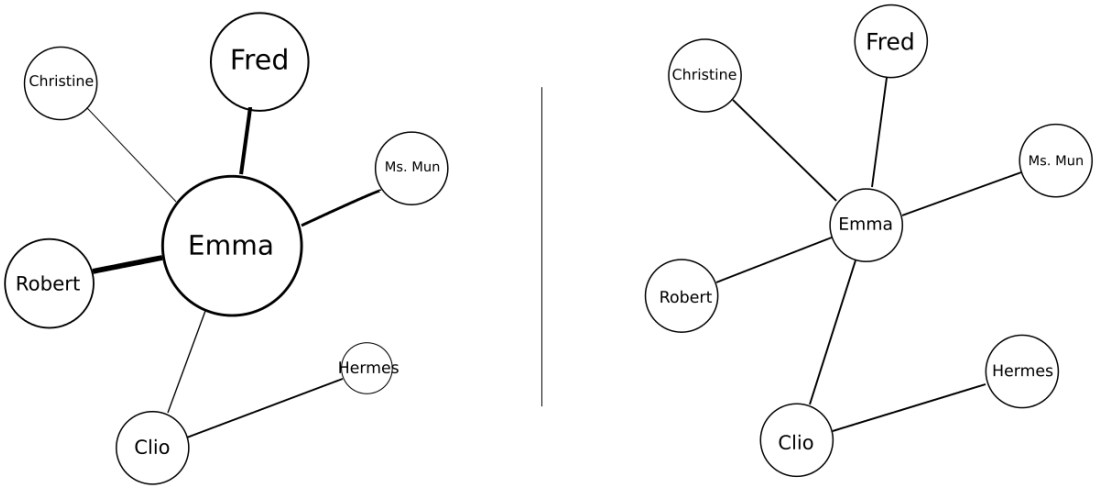

Figure 1: Representations output by social network extractors (left) and our extractors (right). Note that social network extractors take nodes as input and output weighted or labeled edges, while ours outputs nodes and binary edges. Our extractors also utilize text entirety, as opposed to dialogue alone.

Our approach to creating these extractors is to use binary classifiers that act on broad sets of automatically-found character candidates and pairs of major characters, to detect salient characters and relationships respectively. We develop a set of global novel-wide features, such as candidate count and coocurrence, and a novel automated labeling method where we read Sparknotes [1], an online reference with character lists and descriptions for literature, and match their data to ours.

2 Related Work

2.1 Named Entity Recognition

Our first problem of character extraction is closely related to NER problems. The goal in NER tasks is to label text token sequences as special entities with respect to the rest of the text. Similarly, in character extraction the goal is to recognize identifiable unique references to major characters, which we take to fall in the class of consecutive token sequences.

As such, we can look to previous work on NER problems to influence our design. A large portion of NER systems attempt to label special entities in short, stand-alone documents based on local features for token sequences, such as capitalization, prefixes and suffixes [16]. These types of word-level features are important in our case as well, but given the basic hypothesis that the salience of characters is dependent on how they appear throughout the book, we can imagine that they would not prove particularly effective.

On the other hand, NER systems which target longer types or sets of documents or attempt to focus on salient entities also exist, and use additional features to significantly boost performance. Previous work in this vein includes systems which target extraction of important figures from news articles [19] and identification of proper nouns in Chinese text [6]. The systems described utilize features such as candidate entity frequency and presence of other entities in vicinity to help capture the comparative importance of desirable entities. Given that in character extraction we aim to identify *salient* characters, these works significantly influence our own.

2.2 Social Network Extraction

Previous character-based attribute extraction work has focused on social network extraction. Social network extraction consists of tasks similar to the relationship labeling task we focus on, but which focus on a representation influenced only by dialogue in novels. The tasks in social network extraction tend to be considerably more well-defined than those we explore - in [8], the authors identify the amount of dialogue between characters to represent the network, and in [2] the authors identify other determinstic features for dialogue such as point-of-view in narration. While the language processing can be difficult to determine these features, they are fairly clear to human annotators.

In contrast, in both our character extraction and relationship identification tasks, we identify *salient* characters and relationships. This concept of *salience* is ambiguous even to many literary scholars; it is often difficult to agree on which characters and relationships are important in novels due to novel complexity, and context which each reader brings with them. We note that ideally, the tasks we wish to address would be satisfied by tools that adapt to different users' perspectives, possibly taking into account personal annotations. In this work, we attempt to overcome this problem by using a source which has aggregated many perspectives, Sparknotes [1], but it is worth noting that given the novel language isn't enough

to fully define *salience* for humans, there is extra complexity in the language processing task.

Despite this difference, it is not to say that the approaches to social network extraction are unrelated to our work. The features used in models for dialogue detection and dialogue attribute detection, such as number of speakers, can potentially be useful in our tasks as well. We make use of some, like cooccurrence features, but there are many that we have not that could be tested in future work.

3 Methodology

Our approach is to treat both character extraction and relationship identification as binary classification problems. We thus have two main stages for our system: a training stage and an execution stage. In the training stage, we collect data, extract features, apply labels, and then train classifiers; in the execution stage we run our extractors on raw test novels, extracting unlabeled data with features and running the classifiers on it. Pipelines for both of these stages are shown in the figure below. In the following sections, we discuss the components of both pipelines in further detail.

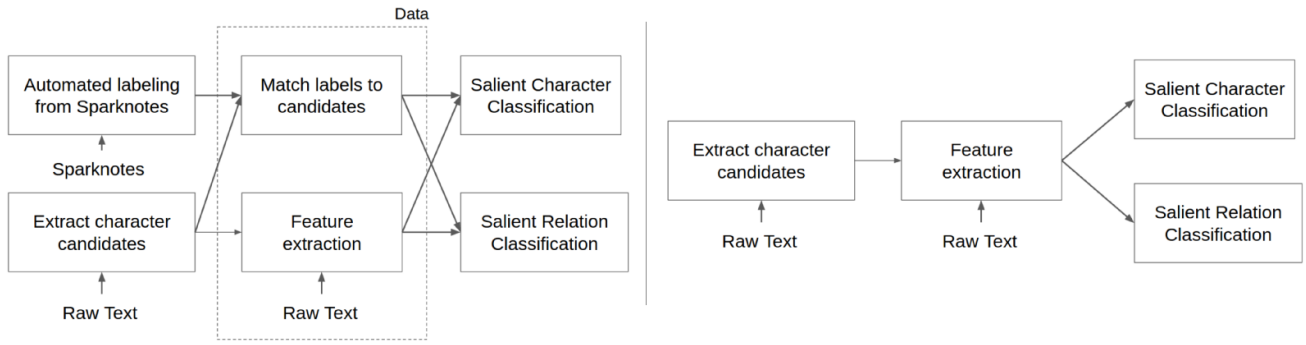

Figure 2: The training (left) and execution (right) pipelines for both our character extractor and relationship identifier. Note that in relationship identification, the training data consists of pairs of candidates labeled as characters given Sparknotes.

3.1 Disambiguation

While not described as a major component in the pipeline, disambiguation is a significant problem at many stages of our work:

1. *Feature extraction:* Disambiguation is used to assign coreference features, letting candidates account for features of coreferences
2. *Labeling:* Disambiguation is necessary to match character names to candidate names
3. *Evaluation:* Disambiguation resolves multiple instances of same-character output from classifiers

For this reason we describe how we deal with it separately before going into the pipeline blocks.

We use rule-based disambiguation to resolve multiple different mentions of the same character. The reasoning behind using a rule-based approach rather than a learning approach is twofold: we felt that our candidate disambiguation tasks were simple enough that a rule-based approach could account for variation well, and we did not have training data for our candidate disambiguation tasks. The latter meant we would either need labeling time which was unavailable, or to use a pre-existing model which was not fitted to our data.

The disambiguation rules can be split into three broad categories:

- *Partial reference* - resolves mentions of characters by partial mentions of their full name, e.g. "Huck" referring to "Huck Finn". This is accomplished by checking containment or partial containment of a name within another.
- *Nickname resolution* - resolves mentions of characters by nickname, e.g. "Huck" as "Huckleberry". This is accomplished by a combination of fuzzy string matching, prefix similarity, and partial containment.
- *Title resolution* - resolves mentions of characters named by title to their full name, e.g. "Mr. Finn" to "Huckleberry Finn". For this, we used a large name-gender database extracted from Stanford's CoreNLP library to decide which names various gender dependent titles could map to, and then look for partial containment in the remaining strings.

We view candidate names as nodes in a graph, and disambiguated a candidate name to another as directed edges in the graph. To enforce transitivity of disambiguation (i.e. if A maps to B and B maps to C, then A maps to C), we then insert additional edges from each node to all reachable other nodes. This is implemented as a sequence of depth-first searches from each node to determine connectivity and reachability.

3.2 Candidate Extraction

Our classifiers, rather than run on all token sequences in text, are both trained and executed on broad sets of multi-token character *candidates* obtained from the raw text automatically. As in certain multi-token NER systems [7], this strategy is employed because we do not know a priori the number of tokens for characters and want to avoid training on large numbers of clearly negative examples. In addition, it is also because the complexity of our feature extraction depends on the number of candidates through co-occurrence counting.

The method for candidate extraction is very tied to our understanding of character names: we assume that character names accord to the following rules:

1. A name is a noun phrases with at most one level of nesting and ending in a capitalized noun. OR

2. A name is a noun phrase consisting of a determiner and a hyponyms "person" in WordNet [12].

We extract all token sequences satisfying these rules as candidates. While many character names are captured just by taking consecutive capitalized tokens, the rules are more flexible to account for character names such as "the Count of Monte Cristo" and "the helmsman", characters in the eponymously-named book and *Heart of Darkness* respectively. In the first we have parts of speech other than nouns, while in the second there is no capitalization, just the recognition that the word refers to a person. There may be characters names which break these rule, but we did not observe any.

This rule-based approach also faces potential issues in that unmeaningful token sequences such as "the woman" in a book with many women may get passed and affect cooccurrence features; however, we observe that cooccurrence with even these "non-characters" is often important, and also include weighted cooccurrence features to help account for this issue. Though we do not report numerical results, we observe that not including these candidates does not affect end performance.

3.3 Feature Extraction

We extract four main types of features for both candidates and pairs: tag features, coreference features, frequency features, and coocurrence features. The first three are common for NER systems, and are included due to the tasks' similarity to NER noted previously. The last feature type, also seen in *Related Entity Finding* tasks [5, 18], is to account for the idea that characters are considered salient due to their relationships with other entities in the novel, and that cooccurrence of characters in some section indicates interaction.

There are actually a large number of features included for each feature type, arising from a number of normalization schemes we use to account for differences between novels. We do not expect all of these to be particularly helpful, and instead count on our learning methods ruling out noisy or uncorrelated features. The main subclasses of features for each feature type are described below; we avoid detailing each particular feature for space, and because those not described are very similar to at least one of the subclasses described.

3.3.1 Tag Features

What we call *tag* features refer to features dependent on capitalization of tokens in candidates, and whether Stanford's CoreNLP NER tagger [9] labeled tokens as named entities or not. We noted that while current NER systems such as Stanford's were not very good at understanding when titles or modifiers were parts of names, they frequently recognized given names of characters, like "Sally". Since character names often include given names, we decided to use the presence of NER tags for indicator features.

As we need to normalize the tag features we obtained over candidates, sequences of tokens, we use presence of capitalization or NER "person" tag on last token, as well as fractions of

tokens with capitalization or "person" tags. The NER tagger is non-deterministic so we use a fractional count of "person" tags over all instances of the candidate for NER tag features. Tag features for pairs of candidates are just concatenations of the tag features for both candidates.

3.3.2 Frequency Features

Frequency features are indications of how often a candidate appears in the given novel. We include these since we expect salient characters and relationships to be tied to how often they occur. The simplest feature of this type is the raw frequency of how many times a candidate appears in the novel. However, we include many variations on this feature to account for differences in book lengths and differences in general candidate reference frequency across books. These include frequencies for candidates across different section types (sentence, paragraph, and chapter), as well as normalization across the number of sections and total number of section mentions for candidates. Explicitly, for section type s, we have:

$$\text{count}_s(\text{candidate}) = (\#\text{ of sections candidate appears in})$$

$$\overline{\text{count}}_s(\text{candidate}) = \frac{\text{count}_s(\text{candidate})}{\#\text{ of sections}}$$

$$\hat{\text{count}}_s(\text{candidate}) = \frac{\text{count}_s(\text{candidate})}{\sum_{\text{cand}\in C}\text{count}_s(\text{cand})}$$

The count over the entire book we normalize by the number of characters (text) in the book, and the total counts of all candidates. Again, for pairs of candidates we concatenate the frequency features for both candidates.

3.3.3 Cooccurrence Features

Cooccurrence features describe the number of times candidates appear with each other candidates. While it seems clear that such features would be important for identifying salient relationships, that it would be helpful for salient character extraction is less obvious. Our reasoning is that characters in novels are often important precisely because they interact with other characters.

The particular features of this type we compute are similar to our frequency features. We identify which sections characters occur in to build occurrence matrices, and then compute inner products of these occurrence matrices to obtain coocurrence matrices. The features for candidates are given by sums of coocurrences with all other candidates.

$$\text{cooc}_s(\text{candidate}) = \sum_{\text{c'}\in C}\text{cooc}_s(\text{candidate}, \text{c'})$$

To normalize across books, as well as try and account for importance of characters, we also weight the cooccurrence matrices along 1 dimension by normalized overall and section frequency features. We do not weight across the second dimension (the one not summed)

because we desire to get the importance of candidates a candidate cooccurs with, not accounting for the importance of the original candidate itself.
For pairs of characters, we both concatenate the coocurrence features for each character, as well as include the values from the original un-marginalized coocurrence matrices.

3.3.4 Coreference Features

Coreference features are to account for cases where characters are referred to by different names within a book. For example, "Huckleberry Finn" may be addressed as "Huck", "Huck Finn", or "Huckleberry", but all are instances of the same character. As we are using different types of frequencies (including cooccurrence) to help account for importance, to help with measuring a candidate's true importance it should have features recognizing the frequencies of coreferences.

We use our rule-based disambiguation to then compute these features. For each candidate we find the candidates that are disambiguated to, and candidates that disambiguate to it (the difference between "Tom Sawyer" → "Sawyer" and "Sawyer" → "Tom Sawyer"), then sum the different frequency features for candidates from those two types of coreference.

3.4 Data Collection

The goal of the data collection phase is to compile a collection of raw texts and Sparknotes character annotations for as many books as possible. To this end, data collection is divided into multiple phases:

1. Gather a list of books with character descriptions available on Sparknotes
2. Crossreference the list with Project Gutenberg [10] to optain books with public, online text
3. Extract candidates from the obtained raw texts
4. Label the candidates for each novel by using the Sparknotes character descriptions

For the first, we scrape the index on Sparknotes, then use the index to find links for each book in Sparknotes and scrape their character lists, title and author. For the second, we search for title and author in the Project Gutenberg index, downloading the raw texts for matches. In practice, there are a number of inconsistencies between Gutenberg's index and Sparknotes', such as differences in accents or spelling, as well as non-English same-title duplicates. We manually dealt with such entries, and then filtered out plays and poems by crossreferencing with Wikipedia to get 102 novels.

Candidate extraction, the third step, is performed in the manner described in **Section 3.1**. To label the candidates we use our rule-based disambiguation to find candidates corresponding to the character names in the Sparknotes descriptions. Rather than attempt to match characters uniquely as would be ideal, we label *all* possible matching candidates as characters. While this can introduce false positives, such as with family names being labeled as characters, we found this significantly improved performance. The classifiers were less strict

about using particular character references, which were apparently not consistent in feature space across different books in Sparknotes (some lists used short names, while some used formal, without apparent reason). We attempt to mitigate false positives by performing a disambiguation after output.

For relationship labeling, Sparknotes does not provide a list of relations, and we instead infer it from character descriptions. The strategy is simple - we look for appearances of a character name within other characters' descriptions, and label them as related if they appear. While character names can appear in different forms, we found that we could perform disambiguation for this stage accurately by just choosing the first-appearing character in the list which our rule-based disambiguation matched. We checked the performance of this labeling strategy on a small set of 40 books we tagged manually from reading character lists ourselves, and achieved roughly 95% precision and 80% recall. Though not accurate, and suffering, as we note later, from not being descriptive of many relationships we personally found important in novels we knew, we proceeded to use this data and see what results could be obtained.

3.5 Classification

We employed two classification methods: SVM and random forests. These two methods have been shown to have been shown to perform well for a variety of classification problems, are particularly useful to avoid overfitting [4, 11], and were accessible through Python's `scikit-learn`. We expected that 100 novels was not enough to represent the novel space very completely, making overfitting avoidance more important.

We also note that our datasets, both for characters and relationships, are imbalanced. As the methods we use can be susceptible to dataset imbalance, we use the method of data weighting for correction. As it was not clear what factor would be ideal to reweight the positive examples, we treated it as a hyperparameter. For the character extractor, we found that weights between $2 - 4$ gave similar and best performance, while with the relationship identifier 10 worked best.

Lastly, we emphasize that while the character extractor is trained on all candidates, since our relationship identifier is meant to identify relationships between salient characters, we train it only on pairs of characters which were labeled as positive data for character extraction.

4 Experiments

We tested our classifiers with both quantitative and qualitative metrics. The quantitative metrics we used included precision and recall. Because our classifier is trained with positive labels on all matching candidates found with our disambiguation, we expect that our classifiers will output multiple references to the same character. Thus, we include a "unique" precision (UP) and recall (UR) as well, where output characters are first disambiguated, and

then matched to the pure Sparknotes characters.

The qualitative metrics included visual verification of characters found by the classifiers for test novels. We avoid showing qualitative metrics for our relationship identifier due to the clearly poor performance we observed - visually printing relationships found did not appear to give us any information on the poor performance. For all our classifiers mentioned in this section, we used a training data ratio of 0.7.

4.1 Character Extraction

For character extraction we implemented one simple baseline and two versions of our method. The baseline was extraction of candidates by frequency: we took the top 20 1-gram candidates, 10 2-gram candidates, and 5 3-gram candidates. While precision and recall will vary between different types of novels, our choices for these numbers stemmed from the fact that Sparknotes had an average of 15 candidates per novel, and that we had an average of 15 candidates after disambiguation with these numbers. Characters also, we found, are typically referred to by shorter identifiers, leading to the decreasing numbers with respect to n-gram length.

The two versions of our method are differentiated in feature sets: for Version 1 we only used frequency features, while for Version 2 we used the entire feature set. We tested both of these versions as we were interested in seeing if features besides frequency really were helpful in the character extraction task. A table describing the results of all 3 methods is shown below.

Method	**Precision**	**Recall**	**Unique Prec.**	**Unique Recall**
Baseline	0.347	0.486	0.351	0.508
Version 1 (SVM, RBF)	0.642	0.544	0.638	0.572
Version 1 (RF)	0.619	0.441	0.615	0.497
Version 2 (SVM, RBF)	0.664	0.681	0.684	0.593
Version 2 (RF)	0.756	0.490	0.745	0.511

Table 1: Test precision and recall for our character extractor.

We notice first that both variations of our method significantly outperform the baseline, regardless of the classification method used. This indicates that the classification approach is useful, since the various features can be used for discrimination, and the high dimensionality of the feature space makes rule-based approaches difficult. On the other hand, the difference in performance between Version 1 and Version 2 only shows in non-unique precision and recall, indicating that feature types besides frequency are not particularly helpful when included. While slightly counterintuitive, it appears that candidate selection already provides the information of tag features, coocurrence features are not much different than frequency features, and that coreference features do not stop at least one of the true character references passing the classifier.

The overall performance of our approach, while promising however, does not appear high enough to warrant adoption for literary purposes, with precision and recall both roughly 65%. It seems that the features chosen are still not expressive enough for this task, or that the derived features from our kernels are not expressive enough.

However, we recognize another source of error from looking at the characters output by our character extractors. Here we show *Crime and Punishment* output for example:

True: Katerina Ivanovna, Sonechka, Lebezyatnikov, Zossimov, Alexander Zamyotov, Dmitri Razumikhin, Lizaveta, Seymon Marmeladov, Ilya Petrovich, Luzhin, Arkady Svidrigailov, Polya, Mikolka, Nastenka, Alyona Ivanovna, Raskolnikov, Porfiry Petrovich, Pulcheria

Predicted: Zossimov, Lizaveta, Raskolnikov, Svidrigailov, Nastasya, Razumikhin, Nikolay, Lebeziatnikov, Sonia, **Dounia**, Zametov, Luzhin, **Amalia Ivanovna**, Sonechka, Porfiry Petrovich, Katerina Ivanovna, Pulcheria, **Marfa Petrovna**, Ilya Petrovich

The characters output by our classifier do appear to be considerably important to their respective novels, it is just that Sparknotes does not label them in their character lists (or has alternate spellings we miss). It is difficult to say if this is just because truly aggregating sources shows these characters are not significant, or if the aggregation that Sparknotes has is not properly representative, as some books have what we see as very obvious characters left out of Sparknotes' salient character list. This is the issue we noted in our data collection when comparing the automatically collected data with manual tags.

4.2 Relationship Identification

For relationship identification we also attempted to implement one simple baseline and two versions of our method. The baseline was analagous - instead we used pairs of characters whose coocurrence in paragraphs crossed a manually-tuned threshold (0.1, normalized by number of paragraphs). Notably though, it does not limit the number of relationships identified, unlike the character extractor. This accords with a phenomenon we saw on Sparknotes: no matter how many characters there are, roughly the same number are recognized as salient for each book; relationships are not like this.

The two versions of our method are again differentiated in feature sets: for Version 1 we only use pairs' normalized coocurrence for paragraphs, while for Version 2 we use the concatenated individual candidate features as well. Unfortunately, given the large number of training samples (on the order of hundreds of thousands), we were not able to train the SVM, which we saw to give better results in the character extraction. Results for our baseline and the first feature set are shown in table below.

Method	Precision	Recall
Baseline	0.303	0.198
Version 1 (RF)	0.144	0.392
Version 1 (SVM, RBF)	0.188	0.994

Table 2: Precision/Recall for the relationship identifier. Note that results for the models with all features is unreported due to unfinished training.

Our recall is particularly high in cases because of the high bias towards positive examples. We noted that if we reduced this high bias, recall and precision both moved to slightly above the baseline. The trained result shown gave a much better F-score. We attempt to explain our issues in the following paragraphs.
It would seem from the results that coocurrence is not a defining enough feature type to identify salient relationships. Our baseline achieves very low precision and recall, and while our trained classifiers perform better, they also have poor performance. We may need to add entirely different feature types, or take significantly different approaches.

On the other hand, we also note that for the same data issues with the character extractor, the relationship identifier has problems as well. In fact, when we compare our own personal knowledge of novels with the relationships obtained from Sparknotes, it is frequent that Sparknotes does not identify a large number of relationships which seem clearly salient in just the character descriptions. For example, in the *Lord of the Rings* novels, relationships between Frodo and nearly half of the Fellowship are unlisted, when they all spend the entirety of the novel traveling together for a common purpose. Since we find that this issue is significantly more egregious for relationships than characters, we expect that it is important to explaining the poor classifier performance, and also the particular precision/recall trade-off we see.

5 Conclusion

In this paper we have motivated the problems of salient character extraction and relationship identification from raw novel text, and described a supervised classification-based approach to solving them. The features we use are very similar in spirit to those in NER systems, and our labeling is made feasible due to automated scraping from a source with simply-parseable data. We find that the approach gives promising performance for character extraction, even with possible issues in labeled data, capturing importance particularly with frequency features; however, the performance for relationship identification is poor, indicating significant issues with labeling via Sparknotes or lack of expressiveness in the features described. The results indicate many directions for future work: improvement in labeling by aggregating different resources such as Wikipedia, better feature sets via design or by learning with neural networks, and structured prediction are all directions we would like to explore.

References

[1] Sparknotes. `http://www.sparknotes.com/`. Accessed: 2015-11-15.

[2] A. Agarwal, A. Corvalan, J. Jensen, and O. Rambow. Social network analysis of alice in wonderland. In *Workshop on Computational Linguistics for Literature*, pages 88–96, 2012.

[3] M. M. Bakhtin. Forms of time and of the chronotope in the novel: Notes toward a historical poetics. *Narrative dynamics: Essays on time, plot, closure, and frames*, pages 15–24, 1937.

[4] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[5] M. Bron, K. Balog, and M. d. Rijke. Related entity finding based on co-occurrence. Technical report, DTIC Document, 2009.

[6] H.-H. Chen and J.-C. Lee. Identification and classification of proper nouns in chinese texts. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 222–229. Association for Computational Linguistics, 1996.

[7] J. F. Da Silva, Z. Kozareva, and J. G. P. Lopes. Cluster analysis and classification of named entities. In *LREC*, 2004.

[8] D. K. Elson, N. Dames, and K. R. McKeown. Extracting social networks from literary fiction. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 138–147. Association for Computational Linguistics, 2010.

[9] J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

[10] M. Hart. *Project gutenberg*. Project Gutenberg, 1971.

[11] M. A. Hearst, S. T. Dumais, E. Osman, J. Platt, and B. Scholkopf. Support vector machines. *Intelligent Systems and their Applications, IEEE*, 13(4):18–28, 1998.

[12] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38 (11):39–41, 1995.

[13] F. Moretti. *Atlas of the European novel, 1800-1900*. Verso, 1999.

[14] F. Moretti. *Graphs, maps, trees: abstract models for a literary history*. Verso, 2005.

[15] F. Moretti. *Distant reading*. Verso Books, 2013.

[16] D. Nadeau and S. Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[17] R. Williams. *The country and the city*, volume 423. Oxford University Press, 1975.

[18] Q. Yang, P. Jiang, C. Zhang, and Z. Niu. Experiments on related entity finding track at trec 2009. Technical report, DTIC Document, 2009.

[19] L. Zhang, Y. Pan, and T. Zhang. Focused named entity recognition using machine learning. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 281–288. ACM, 2004.