CSE 517 - Winter 2015
Final Project Report

Who Did It? "Investigating" Detective Fiction Characters

Abstract

We present a preliminary study of an intriguing problem: predicting a character's *guiltiness* in Agatha Christie's detective stories. Our main goal was to design character representations to be used as features to a guilty/innocent classifier. We experimented with elementary features such as number of mentions and typed dependencies, as well as continuous-space features derived from latent semantic analysis (LSA) and the very recent vector space word embeddings techniques used in `word2vec` and `GloVe`. Our results, while beating an easy baseline in most cases, indicate much more research is needed to make meaningful conclusions.

1 Introduction

Thanks to the increasingly powerful tools that NLP provides, almost any problem involving language can now be studied. In particular, recent years have witnessed many exciting applications in the intersection of NLP and social science/humanities: Feng et al. used syntactic stylometry and distributional patterns to detect deception in product reviews [1, 2], while Ott et al. looked at the imaginative elements of product reviews to detect opinion spams [3]. An abundant source of text, literature has naturally become a rich source for creative NLP tasks. Ashok et al. [4], for example, used statistics of lexical and syntactic rules to predict success of literary works. Elson et al. used automatic quote attributions [5] in character conversations to build a social network in literary fiction, which ultimately provided evidence that surprisingly contradicted several literary theories [6]. Also in the literary domain, Bamman et al. modeled character types in 18^{th} and 19^{th} century novels for several persona comparison tasks, which were evaluated against preregistered literary hypotheses.

It would be intriguing to take a step further and try to predict a story's outcome given an author's writing patterns, such as predicting a detective story's ultimate perpetrator. While a good detective story usually has an unpredictable plot, making these patterns difficult to find, there might however exist potential writing habits that even the author herself is unaware of, whether it is the writing style around certain characters, or the personality traits these characters might all share. The idea of this project is attempting to identify these patterns. The rest of this document is organized as follows: Section 2 describes a recent work most closely related to what we're exploring, the software, and the data we have available. In Section 3, we explain the approaches we explored for our "investigative" task. Section 4 presents our findings and Section 5 proposes possible next directions to which we would expand this project. Finally, concluding remarks are presented in Section 6.

2 Related Work, Software, and Data

To the best of our knowledge, while there have been many works applying NLP techniques to literary analysis, no work has directly addressed the task of perpetrator character detection proposed here. Since our goal is to detect a certain character in the story, the "criminal", we found Bamman et al.'s work [7] in persona modeling most related to our task. Though our goal is not necessarily to model personas, we thought certain aspects of Bamman et al.'s work, in particular how to represent literary characters in topic models, might be applicable to our project.

In [7], Bamman et al. explored the problem of modeling character types, taking into account extra-linguistic information, such as author and writing period, responsible for generating texts associated with these characters. The target of modeling, *persona*, is defined as a distribution over several typed dependency relations (roles) the characters can take on (agent, patient, possessive, and predicative). Each tuple of (role, word) is then assumed to be drawn from a log-linear distribution with the feature set: metadata (author), persona (latent variable), and background (word features), whose parameters are then learned via stochastic EM and Gibbs sampling. For evaluation, the authors used as gold standard a set of 29 hypotheses preregistered by literary scholars. These hypotheses take the form "character X is more similar to character Y than either X or Y is to a distractor character Z". The model was then tested by its ability to confirm or failing to confirm the decisions:

$$\begin{gathered} distance(X,Y) < distance(X,Z) \\ distance(X,Y) < distance(Y,Z) \end{gathered}$$

To scale computations to book-length documents, Bamman et al. constructed a preprocessing pipeline[1], which we found very helpful and therefore adopted as our preprocessing pipleline as well. Similar to the authors, we used this pipeline for essential upstream tasks: POS tagging (Stanford POS tagger [8]), parsing and dependency parsing (MaltParser [9]), and most importantly character clustering (via NER [10] and coreference resolution). In addition, we had several toolkits readily available for learning word embeddings, `word2vec`[2] [11] and `GloVe`[3] [12], which we used as main features in our model (details in Section 3).

The data we used consists of Agatha Christie's detective novels (we have not considered her short stories or plays). Agatha Christie's work is particularly well-suited for this study since most of her stories involve one main criminal revealed at the end. Two texts are available free on Project Gutenberg[4], many others were fortunately donated by an acquaintance. We had in total 48 annotated novels (with characters marked *victim* or *criminal*, or neither), 41 of which were used for training and the rest for testing. The total data size amounted to 4.3 million word tokens, with a vocabulary of 53,000 word types.

3 Methods and Models

Our goal of detecting the main culprit in each story is formulated as a classification task, where each person-entity in a novel is assigned the label "guilty" (positive sample) or "innocent" (negative sample). The focus of our work is therefore to design vector representations of characters (which we refer to as *character embeddings*), to use as features to our classifier.

3.1 Data Preparation

A significant challenge in learning character embeddings is the issue of new, previously unseen characters, much like the issue of unknown words in language models. However, in our case, we cannot simply assign an 'UNK' token to new characters, since this strategy would potentially conflate new *guilty* and *innocent* characters. On the other hand, the true identity of the criminal is often revealed in the last few chapters of each novel, which means we would, to an extent, have access to the true labels were we to include the last portions of each novel in learning character embeddings. With this in mind, we learned all our models from two types of data: *complete* data, where we used all texts available, and *incomplete* data, where we

[1]`http://www.ark.cs.cmu.edu/literaryCharacter/`
[2]`https://code.google.com/p/word2vec/`
[3]`http://nlp.stanford.edu/projects/glove/`
[4]`http://www.gutenberg.org`

excluded the last 15% of each story from our model learning – simulating the effect of not knowing the true culprit until the end of the story. In other words, the *complete* set character embeddings should capture all information about the characters, while the *incomplete* set character embeddings should capture all information up to 85% of the story the character is in. We therefore posit that the results from the *complete* data models would be an upper bound to the quality of our classifier and embedding models.

Using the pipeline provided by Bamman et al. [7], we have processed stories where each token is annotated with its POS tag and typed dependency tag. For named entities and pronouns (both personal and possessive), each token is additionally assigned a character ID, which was found by clustering coreference-resolved entities. We define a character **mention** as any instance labeled with a character ID, thus a *mention* could refer to the proper name (and its aliases), the personal and the possessive pronouns referring to the character. In this work, we did not distinguish between mentions by the narrator and those by a different character; mentions in narration vs. conversations were treated the same as well (though these distinctions would be an interesting direction to explore). Feature extraction and character embeddings are therefore learned from these instances in the text. The example below illustrates what is essentially done to the original text: in effect, we replace each *mention* of the character by the corresponding character ID, annotated with additional information of the story it belongs to.

Mrs. Inglethorp greeted me with effusion. "Why, if it isn't too delightful to see you again, Mr. Hastings, after all these years. Alfred, darling, Mr. Hastings—my husband." I looked with some curiosity at "Alfred darling". He certainly struck a rather alien note. I did not wonder at **John** objecting to his beard. It was one of the longest and blackest I have ever seen. He wore gold-rimmed pince-nez, and had a curious impassivity of feature.

$$\Downarrow$$

CHAR00_STYLES CHAR00_STYLES[5] greeted me with effusion . " Why , if it is n't too delightful to see you again , CHAR19_STYLES CHAR19_STYLES , after all these years . CHAR33_STYLES , darling , CHAR19_STYLES CHAR19_STYLES - - my husband . " I looked with some curiosity at " Alfred darling " . He certainly struck a rather alien note . I did not wonder at CHAR46_STYLES objecting to CHAR46_STYLES beard . It was one of the longest and blackest I have ever seen . CHAR46_STYLES wore gold-rimmed pince-nez , and had a curious impassivity of feature .

The example above illustrates several (interesting) issues: (1) the English honorifics Mrs. and Mr. were treated as separate mentions from the proper nouns they are attached to; (2) the character clustering system missed (or could not resolve) certain pronouns (He certainly struck a rather alien note...) while (3) incorrectly resolved other pronouns (I did not wonder at **John** objecting to his beard... and He wore gold-rimmed pince-nez...). Additionally, not shown here are examples where the coref system could not recognize the same entity with different aliases[6]. At the time of this work, we did not have a good way of handling these preprocessing issues[7].

We describe our character feature learning in the following sections, note that all these models were learned separately for both the *complete* and *incomplete* data. Also, we should clarify that the train/test split in terms of stories (41/7 train/test) is only used for learning the classifier. For the embedding learning, due to 'UNK' character issues described above, we used all stories in either the *complete* or the *incomplete* texts. To be more explicit, we learn the embeddings first, then the characters in each *complete*/*incomplete* set are split into training and testing characters for constructing and evaluating the classifier.

[5]Each mention was converted to the format CHARXX_YYYYYY where XX denotes the character ID and YYYYYY denotes the 6-letter abbreviation of the story this character belongs to. Here the excerpt was taken from *The Mysterious Affair at Styles*.

[6]In one of the other stories, *Lord Edgware Dies*, Edgware's wife's name was Jane Wilkinson, which the system could not recognize (understandably) as the same entity as Lady Edgware.

[7]Bamman et al. also noted that their coreference resolution system achieved 82% accuracy, and coreference resolution is a difficult problem on its own, so this is probably the best we can get for now.

3.2 Baseline and Elementary Features

Our baseline classifier is the random guesser: given a story and its characters, we randomly assign one of the characters the label *guilty*. We also considered two elementary features on their own and combined: (1) the number of mentions associated with each character (normalized) and (2) the dependency tags (as one-hot) vector each mention is annotated with. In particular, we first surveyed our tokenized texts to find the most common dependency tags attached to a character entity (names and pronouns included). We then encoded each mention with a 6-dimensional vector representing the 5 most common dependency tags, reserving an additional dimension for all other tags: $[nsubj, poss, pobj, dobj, nsubjpass, other]$[8]. To represent the dependency features of a character as a whole, we used the sum of all the dependency vectors of the mentions associated with the character.

3.3 Continuous-Space Embeddings of Characters

3.3.1 Latent Semantic Analysis

The first approach to modeling character features in the semantic space is by using LSA. Our LSA matrix was constructed as follows. We first created a fixed vocabulary (which included our CHARXX_YYYYYY notations) and performed a few preprocessing steps: (1) converted all rare words (count < 3, except if the word is one of the CHARXX_YYYYYY tokens) to the 'UNK' token, and (2) converted all numbered words to the 'NUMBER' token. This effectively halved our vocabulary, resulting in 24,000-26,000 word types. Table 1 summarizes our data set characteristics. Again, note that each modeling, processing, etc. step is done twice, once for the *complete* data and once for the *incomplete* data.

We then defined a document as a context window around each character mention. We consider 2 types of windows: 20 words to the left and right of each mention, and 40 words to the left and right of each mention but counting only content words[9]. The term-document matrix was then constructed by tallying accumulated co-occurrence counts of vocab-words in each document. This matrix was then normalized TF-IDF, and finally rank-reduced via SVD to dimension 100, i.e. each document (character mention) is represented by a 100-dimensional vector[10]. Similar to what we did with dependency vectors, we sum the mention-vectors associated with each character to get the vector representation of the character. We will refer to these vector features as "LSA" (for the 20-word context window) and "LSA.content" (for the 40-word context window). Additionally, we also considered adding the normed mentions, the dependency vectors, or both to use as overall features representing each character.

Alternative to using each character-vector as features to our classifier, we considered using each mention-vector as features to the classifier. In other words, instead of summing the resulted LSA/LSA.content rank-reduced vectors, we treat them as feature vectors on their own. The reasoning behind this was that treating each mention-vector as a feature vector gave us more data to learn from, as well as a higher percentage of positive samples, especially since we have a class imbalance situation (see Table 1)[11].

We recognize the fact that these mentions are not independent, therefore also considered performing classification+voting in experiments using mentions. Specifically, we tried thresholding on the fraction of positive mention-classifications to get the final decision on the character's "guiltiness". For a certain character Z, let us denote the number of positive predicted labels as p_Z and negative predicted labels as

[8]The full descriptions of all dependency tags can be found in the Stanford typed dependency parser manual. `http://nlp.stanford.edu/software/dependencies_manual.pdf`

[9]We initially considered using previous/following paragraphs as contexts, but these often resulted in very few words, due to conversations in the story.

[10]We wrote a Python script to construct the matrix, then performed SVD using Python's scipy.sparse.linalg package.

[11]The fact that the percentage of positive samples increased in the case of mention-vectors suggests that the number of mentions positively correlates with positive samples

Table 1: Summary of Data Characteristics

	Complete Set	Incomplete Set
Number of Tokens	4.3 M	3.7 M
Original Vocabulary	52.7 K	49.9 K
Processed Vocabulary	26.4 K	24.6 K
Number of Characters	2,937	2,736
Number of Positive Samples - Characters (percentage guilty)	148 (5.0%)	145 (5.2%)
Number of Mentions	236,437	196,489
Number of Positive Samples - Mentions (percentage guilty)	30,758 (13%)	23,876 (12.2%)

n_Z. Then Z is marked "guilty" if $\frac{p_Z}{p_Z+n_Z} > t$, where t is a threshold we varied between 0 and 0.5 in 0.1 steps.

3.3.2 Other Continuous-Space Word Embedding Tools

Recently, many powerful tools for continuous-space word representations have proven to outperform LSA embeddings in various semantic as well as syntactic tasks, such as `word2vec` by Mikolov et al. [11] and `GloVe` by Pennington et al. [12]. Taking advantage of these readily available toolkits, we could also learn character embeddings in each story by relabeling each character mention in our raw text by the CHARXX_YYYYYY notations. Each of these mentions are then treated as a word in the vocabulary by the embedding toolkits, and we get the character embeddings by training `word2vec` and `GloVe` on our relabeled data. For both `word2vec` (Skip-gram model) and `GloVe`, we set the vector dimension to 100 and the window to the default 11. As with the LSA and LSA.content vectors, we also considered adding mentions, dependencies, and both to get an overall representation of our characters.

4 Experiments and Results

Since the character (mention) vectors are have continuous values while the dependency features have discrete values, we considered two standard classifiers: a logistic regression classifier and a decision tree. We used L2-regularization on logistic regression, where the hyperparameter C was tuned by 5-fold cross-validation on training data. C was varied in the range [1,1000] in 10 logarithmic steps; i.e. $C \in \{1, 2.2, 4.6, 10, 21.5, 46.4, 100, 215.4, 464.2, 1000\}$. For the decision tree, we set the maximum depth to 15 to avoid overfitting (we didn't tune this parameter on the decision tree because of time constraints; in fact, the only reason we considered a decision tree was because initial results with the logit were poor).

The baseline and elementary feature (mentions, dependencies) results are shown Table 2 and in Figures 1, 2. The last 2 columns of Table 2 show the confusion matrix for each complete/incomplete data set; the figures 1 and 2 show the equivalent information in terms of precision, recall, and F1 score. The confusion matrix is displayed the form: $\begin{array}{c|c} TN & FP \\ \hline FN & TP \end{array}$.

The baseline random guesser unsurprisingly performed poorly, both in the complete and incomplete data cases. Using only normed mentions as a feature also did poorly, in both data cases as well as both the logit and tree classifiers. A difference in performance only arises when we add the dependency features:

Table 2: Results on Baseline and Elementary Features

Feature Vector	Classifier	Model Number	Complete Data		Incomplete Data	
Baseline:	N/A	M0.0	381	7	353	6
Random			23	0	20	1
Mentions	Logistic Regression	M0.1,LR	388	0	359	0
	C=1.0		23	0	21	0
Dependencies	Logistic Regression	M0.2,LR	388	0	359	0
	C=1.0		23	0	21	0
Mentions	Logistic Regression	M0.3,LR	388	0	359	0
+Dependencies	C=1.0		23	0	21	0
Mentions	Decision Tree	M0.1,DT	371	17	334	25
	max_depth=15		21	2	20	1
Dependencies	Decision Tree	M0.2,DT	364	24	**340**	**19**
	max_depth=15		18	5	**15**	**6**
Mentions	Decision Tree	M0.3,DT	**368**	**20**	337	22
+Dependencies	max_depth=15		**17**	**6**	15	6

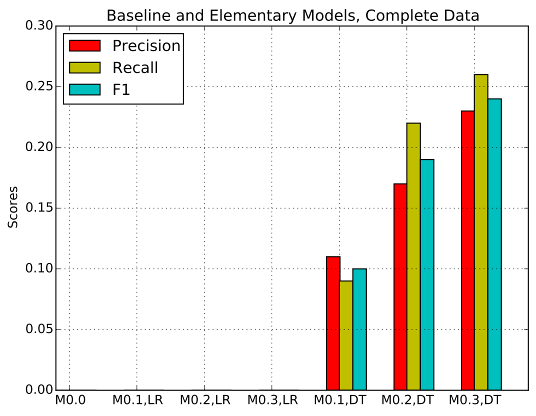

Figure 1: Baselines, Complete Data

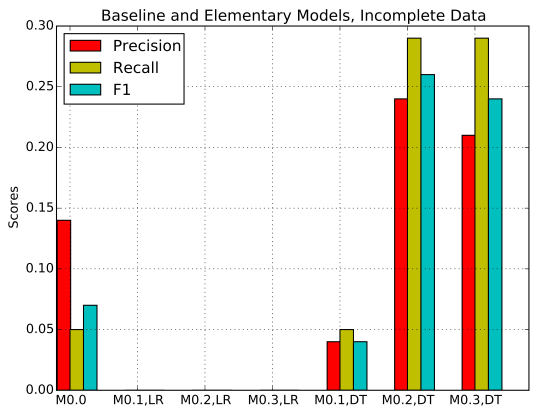

Figure 2: Baselines, Incomplete Data

while the logit classifier still didn't beat the baseline, the decision tree showed some improvement in terms of precision, recall, and F1 score in both the complete and incomplete data sets. While the absolute score values are slightly higher in the incomplete data than in the complete data, interestingly, the confusion matrix values suggest that this difference is probably not meaningful.

Similar results can be seen in Figures 3 through 6, which show the performance of LSA models (the corresponding confusion matrices can be found in Tables 5 through 6 in the Appendix). Models M1.1 through M1.4 are those using LSA embeddings, LSA+mentions, LSA+dependencies, and LSA+both, respectively. Models M1.1b through M1.4b are defined similarly, except embeddings are the LSA.content ones. In both the complete and incomplete data sets, while the logit classifier performs slightly better with the content embeddings (Models M1.1b through M1.4b), the decision tree results suggest otherwise. Again, it is surprising that, in terms of F1 score, the LSA-logit results are higher in the incomplete data than in the complete ones; this is not the case for the decision tree. However, the confusion matrices imply these differences aren't too significant either, with the best true positive retrievals of 1 sample (complete) and 2 samples (incomplete) for the logit classifier; 4 samples (both complete, incomplete) for the decision tree.

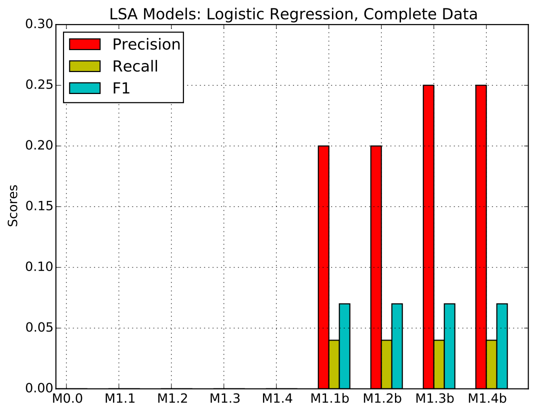

Figure 3: LSA:logit, Complete Data

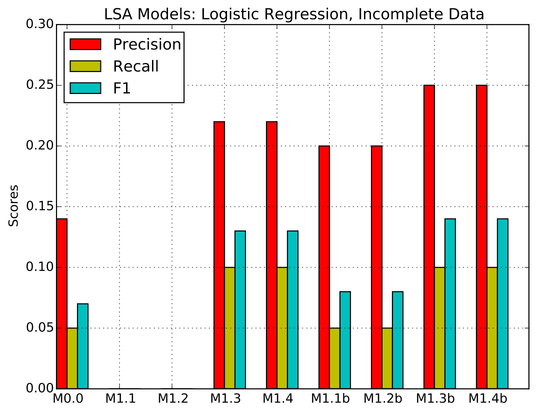

Figure 4: LSA:logit, Incomplete Data

The results of classification using `word2vec` and `GloVe` embeddings are shown in Figures 7 through 10. Models M2.1 through M2.4 are those using `word2vec` embeddings, with and without mentions and dependency features. Models M3.1 through M3.4 are the analogous models using the `GloVe` embeddings. In this case, the logit classifier seems to perform better than the decision tree across all models in the complete data set, however not so much in the incomplete data set.

Compared to LSA, these `word2vec` and `GloVe` results look slightly better, when the classifier got any true positives at all. Again, however, we admit that the differences might not be significant, since the highest number of true positives found across all cases remains at 6 samples for the complete data, and 4 samples for the incomplete data. In fact, this is the same number of true positives retrieved in the best-case elementary feature configuration (mentions+dependencies only) using a decision tree. Therefore, in terms of F1 score, the LSA, `word2vec`, and `GloVe` all yield comparable performance to using only mentions+dependencies as features. However, the precision was highest with `word2vec` embeddings alone in the complete data set and with `GloVe` +mentions+dependencies in the incomplete data set.

Finally, we experimented with using mention-vectors individually as features to the classifier. The confusion matrix results are shown in Table 3. Model M4.1,LR denotes using each LSA mention as a feature

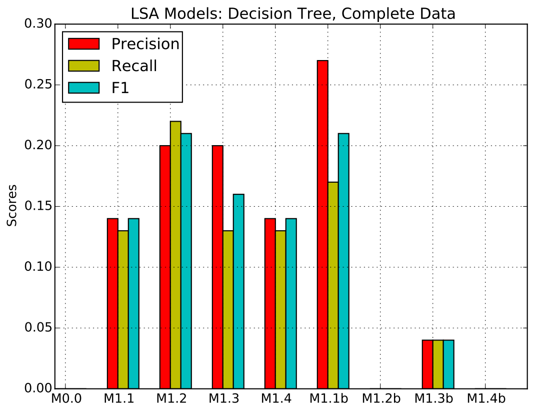

Figure 5: LSA:tree, Complete Data

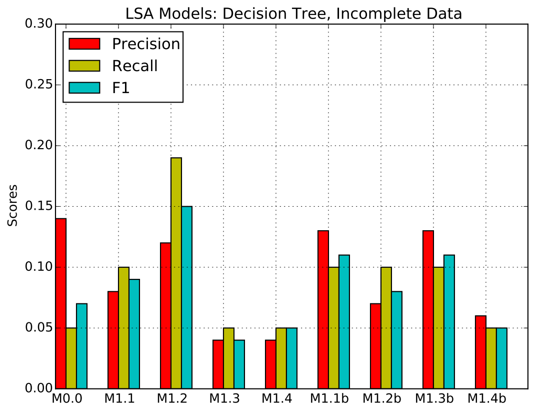

Figure 6: LSA:tree, Incomplete Data

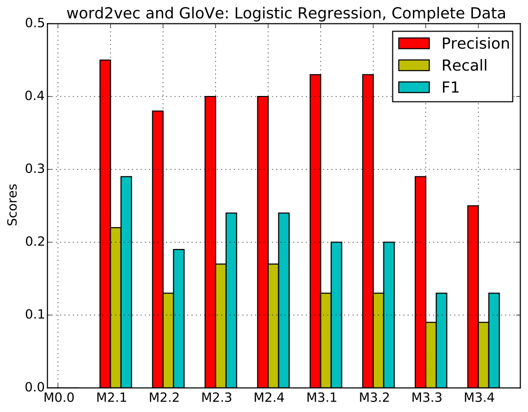

Figure 7: w2vec & GloVe: logit, Complete Data

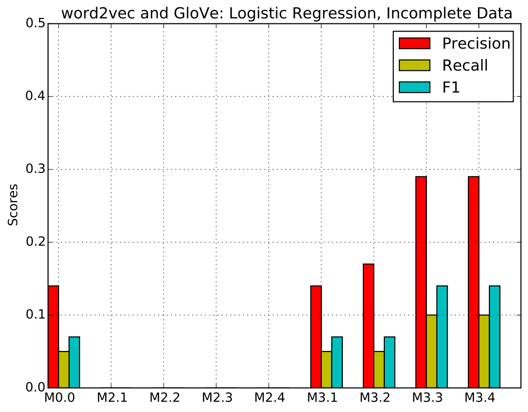

Figure 8: w2vec & GloVe: logit, Incomplete Data

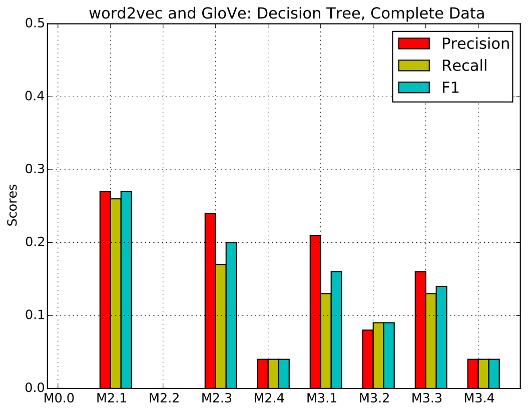

Figure 9: w2vec & GloVe: tree, Complete Data

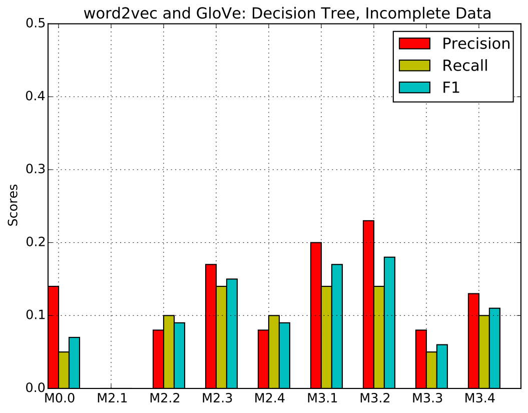

Figure 10: w2vec & GloVe: tree, Incomplete Data

Table 3: LSA Results, Classification with Mentions

Feature Vector	Classifier	Model Number	Complete Data		Incomplete Data	
Baseline:	N/A	M0.0	381	7	353	6
Random			23	0	20	1
LSA	Logistic Regression	M4.1,LR	27,537	3	23,780	0
	C=1.0; C=1.0		5,287	0	3,699	0
LSA	Decision Tree	M4.1,DT	25,330	2,210	22,006	1,774
	max_depth=15		4,784	503	3,366	333
LSA	Decision Tree	M4.1,vote	180	208	159	200
	max_depth=15; t=0		4	19	4	17
LSA	Logistic Regression	M4.2,LR	27,537	3	23,780	0
+Dependencies	C=1.0; C=1.0		5,287	0	3,699	0
LSA	Decision Tree	M4.2,DT	25,391	2,149	21,936	1,844
+Dependencies	max_depth=15		4,778	509	3,369	330
LSA	Decision Tree	M4.2,vote	175	213	165	194
+Dependencies	max_depth=15; t=0		4	19	5	16
LSA.content	Logistic Regression	M4.1b,LR	27,536	4	23,780	0
	C=1.0; C=1.0		5,287	0	3,699	0
LSA.content	Decision Tree	M4.1b,DT	26,078	1,462	21,722	2,058
	max_depth=15		4,963	324	3,329	370
LSA.content	Decision Tree	M4.1b,vote	219	169	**167**	**192**
	max_depth=15; t=0		4	19	**4**	**17**
LSA.content	Logistic Regression	M4.2b,LR	27,535	5	23,780	0
+Dependencies	C=1.0; C=1.0		5,287	0	3,699	0
LSA.content	Decision Tree	M4.2b,DT	26,139	1,401	21,732	2,048
+Dependencies	max_depth=15		4,979	308	3,332	376
LSA.content	Decision Tree	M4.2b,vote	**219**	**169**	165	194
+Dependencies	max_depth=15; t=0		**2**	**21**	5	16

vector with a logit classifier. Again, the L2-regularization parameter was found by 5-fold cross validation; in this experiment all configurations yielded best $C = 1.0$, for both complete and incomplete data. Similarly, model M4.1,DT denotes using LSA mention-vectors as feature vectors with a decision tree, max depth 15. Model M4.1,vote denotes using LSA mention-vectors as feature vectors with a decision tree as well, but classifications on mentions are combined to get a final decision on the character. As described in Section 3, the threshold on the fraction of positive mentions was varied from 0 to 0.5; the table shows only the results of the threshold yielding best F1 scores (more figures and tables regrading this experiment can be found in the Appendix).

Interestingly, we always got the best F1 score if we set the threshold t to 0; i.e. the best strategy seems to be deciding the character is "guilty" if any of its mention was classified as positive. Unsurprisingly, this time our recall increased significantly, up to 0.91 in the complete data and 0.81 in the incomplete data, though precision suffered in return. In terms of F1 score, however, this strategy did not bring any win compared to previous models using one vector per character, with the highest score of 0.2 in the complete data and 0.15 in the incomplete data, both using the LSA.content embeddings.

Models of 4.2-varieties are defined analogously; here the mention-vectors are concatenated with the mention's dependencies. Similarly, models of M4.1b- and M4.2b-varieties also alternate between using mention-vectors and mention-vectors with dependencies, but the embeddings were from the LSA.content configuration. Note that we only did the voting scheme for the decision tree classifier, because unfortunately the logit consistently yielded 0 scores in all cases.

Overall, the highest F1 score across all models was achieved with `word2vec`-only embedding for the complete data (0.29), and with dependencies-only features in the incomplete data (0.26).

5 Future Directions

Before accepting that there aren't any patterns pointing to the ultimate criminal[12], we think the following directions are worth exploring.

5.1 Modifying the Classifier

We have not yet tried tuning our decision tree classifier, we could try varying the maximum depth or maximum nodes allowed in the tree, again using cross validation to select the best parameter. Alternatively, we could explore a different type of classifier, suitable for both continuous and categorical features, such as a support vector machine. In this work, we didn't use an SVM because we would have had to tune not only regularization parameters but also kernel types, which time did not allow. The main problem with our poor results, however, most likely lies in the way we represent characters, especially since even the *complete* data results weren't good. We therefore would like to explore other options of designing character embeddings, as outlined below.

5.2 Modifying Character Representations

As illustrated in the example of Section 3, our character mentions are not of the best quality due to coreference resolution issues. We could avoid this upstream problem by defining mentions as only named entities (i.e. excluding pronouns), but this strategy would severely decrease the amount of data available to us. In fact, Bamman et al. [7] reported that over 70% of references to characters in books are in the form of pronouns. Therefore, we could try looking for a better coreference resolution system, or find a larger data source to learn from. In this work, we wanted to keep the author constant, so we have not

[12]... or celebrating the fact that Agatha Christie was really good at what she did.

yet looked into other authors' detective fiction novels. The variability in authorship might then be a worthwhile tradeoff for our data sparsity problem.

Additionally, we could experiment with treating mentions of characters differently depending on who is referring to them or depending on whether the mention appears in a conversation/narrative. Alternatively, we could study the effects of varying the window sizes in LSA, `word2vec`, as well as `GloVe` training.

Another aspect we haven't had a chance to explore is character appearance order, and how each character develops over the course of the story. We provide some figures of character timelines (in terms of their mentions) in the Appendix, though so far we haven't seen any interesting patterns we could study in more detail.

Yet as another alternative, we could explore a completely different direction by representing characters in terms of their relations to each other. This approach was inspired by the word analogy task with the vector offset method described by Mikolov et al. [13]: to answer the analogy question $a : b \rightarrow c :?$, one first looks at the corresponding vector space representations w_a, w_b, and w_c and computes $y = w_b - w_a + w_c$. The vector y is then the continuous representation most likely to answer the analogy question, so the word is chosen by $d* = \arg\max_d \frac{w_d y}{||w_d||||y||}$; i.e. choose the word whose vector representation has the highest cosine similarity with y. We hypothesize that we can similarly find the criminal of a story by performing the analogy task $victim_1 : criminal_1 \rightarrow victim_2 :?$.

Table 4 shows results of initial explorations in this direction. In particular, we chose the pair ($victim_{STYLES}$: $criminal_{STYLES}$) as the pivot pair ($victim_1 : criminal_1$)[13]. For each new victim $victim_x$, we look at the characters' vector space representations $w_{victim_{STYLES}}$, $w_{criminal_{STYLES}}$, w_{victim_x} and compute the quantity $y = w_{criminal_{STYLES}} - w_{victim_{STYLES}} + w_{victim_x}$. We then iterate over characters in the same story as $victim_x$, and decide guilty the character whose vector representation has the highest cosine similarity with y. We actually retrieved the top k (10, 5, and 2) most "suspicious" characters in terms of cosine similarity, and computed how often the correct criminal appeared in top k, on average (Average Precision @k) for each of the LSA, `word2vec`, and `GloVe` embedding. The Discounted Cumulative Gain (DCG) for each top k retrieved results was computed as in [14]: $DCG_k = \sum_{i=1}^{k} \frac{2^{rel_i}-1}{log2(i+1)}$ where $rel_i \in \{0, 1\}$ with 1 being relevant and 0 otherwise.

Table 4: Results of the Vector Offset Experiment

	Complete Data			Incomplete Data		
	LSA	word2vec	GloVe	LSA	word2vec	GloVe
Average Precision @10	0.62	0.51	**0.70**	0.35	**0.54**	0.38
Average DCG in top 10	0.33	0.34	**0.43**	0.22	**0.35**	0.20
Average Precision @5	0.37	0.37	**0.42**	0.26	**0.33**	0.18
Average DCG in top 5	0.23	0.25	**0.27**	0.18	**0.22**	0.11
Average Precision @2	0.13	0.13	**0.18**	0.11	**0.14**	0.05
Average DCG in top 2	0.10	0.09	**0.13**	0.10	**0.11**	0.05

As can be seen in Table 4, `GloVe` again seems to perform better than others in the complete data set, while `word2vec` does better in the incomplete data set. Obviously, this was just a very naive exploratory experiment; we plan to study more principled approaches of relation extraction in the future. This venue of

[13] As before, STYLES denotes the story *The Mysterious Affair at Styles*. The choice of this particular story was arbitrary.

relation extraction has the advantage of being relatively well-studied so far (i.e. there is enough literature on this topic), but on the other hand would be much more involved than our simplistic and ready-to-use toolkit approaches.

6 Conclusion

In this work, we studied Agatha Christie's detective novels in an attempt to determine the ultimate criminal in a story by experimenting with a few NLP methods. In particular, we designed *character embeddings* to be used as feature vectors to a logistic regression classifier and a decision tree classifier, which learn to decide whether a character is "guilty" or "innocent". For the character embeddings, we experimented with using rank-reduced LSA representations, as well as `word2vec` and `GloVe` trained embeddings. All these continuous embedding vectors were then augmented with the number-of-mention and dependencies, to study the effects of these additional elementary features on the classifier performance. Our results so far are quite poor, with the highest F1 score of 0.29 achieved with `word2vec` embedding in the complete data set, and 0.26 with dependency-only in the incomplete data set. Since we consider the results on the complete set as our upper bound, these results, while disappointing, are perhaps not surprising. Finally, we presented potential next steps to improve our model and hopefully get more sensible results, with the relation extraction being the most promising venue. Overall, we are still very much intrigued by this problem and are excited to develop better approaches to solve it.

References

[1] Song Feng, Ritwik Banerjee, and Yejin Choi. Syntactic Stylometry for Deception Detection. In *ACL (2)*, pages 171–175. The Association for Computer Linguistics, 2012.

[2] Song Feng, Longfei Xing, Anupam Gogar, and Yejin Choi. Distributional Footprints of Deceptive Product Reviews. In John G. Breslin, Nicole B. Ellison, James G. Shanahan, and Zeynep Tufekci, editors, *ICWSM*. The AAAI Press, 2012.

[3] Myle Ott, Yejin Choi, Claire Cardie, and Jeffrey T. Hancock. Finding Deceptive Opinion Spam by Any Stretch of the Imagination. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 309–319, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[4] Vikas Ganjigunte Ashok, Song Feng, and Yejin Choi. Success with Style: Using Writing Style to Predict the Success of Novels. In *EMNLP*, pages 1753–1764. ACL, 2013.

[5] David K. Elson and Kathleen McKeown. Automatic Attribution of Quoted Speech in Literary Narrative. In Maria Fox and David Poole, editors, *AAAI*. AAAI Press, 2010.

[6] David K. Elson, Nicholas Dames, and Kathleen McKeown. Extracting Social Networks from Literary Fiction. In Jan Hajic, Sandra Carberry, and Stephen Clark, editors, *ACL*, page 138–147. The Association for Computer Linguistics, 2010.

[7] David Bamman, Ted Underwood, and Noah A. Smith. A Bayesian Mixed Effects Model of Literary Character. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 370–379, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[8] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich Part-of-speech Tagging with a Cyclic Dependency Network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language*

Technology - Volume 1, NAACL '03, pages 173–180, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[9] Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. Maltparser: A Language-independent System for Data-driven Dependency Parsing. *Natural Language Engineering*, 13(2):95–135, 2007.

[10] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 363–370, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[11] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *CoRR*, abs/1301.3781, 2013.

[12] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics, 2014.

[13] Tomas Mikolov, Wen tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751. The Association for Computational Linguistics, 2013.

[14] www.kaggle.com/wiki/NormalizedDiscountedCumulativeGain Kaggle. Normalized Discounted CumulativeGain.

Appendix

In all the tables following, the confusion matrices are shown in the last 2 columns, corresponding to complete and incomplete data sets. The matrices take the form $\begin{array}{c|c} TN & FP \\ \hline FN & TP \end{array}$.

Results of LSA and LSA.content embeddings using a logistic regression classifier are in Table 5. The L2 regularization parameter C was tuned by 5-fold cross validation; best C values are shown for both the complete and incomplete data cases in the format $C_{complete} = x; C_{incomplete} = y$. Table 6 shows results of LSA and LSA.content embeddings using a decision tree classifier. Maximum depth of tree was set to 15 to avoid overfitting. Table 7 presents results using `word2vec` and `GloVe` embeddings for the logistic regression classifier. The L2 regularization parameter C was also tuned by 5-fold cross validation and best C values are shown as with Tables 5 and 6. Table 8 shows similar results for the decision tree classifier.

Table 3 in Section 4 presented confusion matrices for various LSA and LSA.content configurations using mention-vectors as features. Figures 11 and 12 provide the equivalent information in terms of precision, recall, and F1 scores. In the classification+vote case, the figures only show the best configuration (setting threshold $t = 0$) along side the rest of the models. Figures 13 through 16 show the Precision vs. Recall plots and the F1 vs. threshold plots to illustrate the effects of varying the voting threshold t.

Finally, figures 17 through 22 show several character timeline plots. Red dots represent mentions of the *criminal* while blue dots represent those of the *victim*; black dots represent other neutral characters.

Table 5: LSA Results, Logistic Regression Classifier

Feature Vector	Classifier	Model Number	Complete Data		Incomplete Data	
Baseline:	N/A	M0.0	381	7	353	6
Random			23	0	20	1
LSA	Logistic Regression	M1.1	386	2	357	2
	C=1.0; C=1.0		23	0	21	0
LSA	Logistic Regression	M1.2	386	2	355	4
+Mentions	C=21.5; C=4.6		23	0	21	0
LSA	Logistic Regression	M1.3	386	2	352	7
+Dependencies	C=1.0; C=1.0		23	0	19	2
LSA +Mentions	Logistic Regression	M1.4	386	2	352	7
+Dependencies	C=1.0; C=1.0		23	0	19	2
LSA.content	Logistic Regression	M1.1b	384	4	355	4
	C=1.0; C=1.0		22	1	20	1
LSA.content	Logistic Regression	M1.2b	384	4	355	4
+Mentions	C=100; C=1.0		22	1	20	1
LSA.content	Logistic Regression	M1.3b	385	3	353	6
+Dependencies	C=1.0; C=1.0		22	1	19	2
LSA.content +Mentions	Logistic Regression	M1.4b	**385**	**3**	**353**	**6**
+Dependencies	C=1.0; C=100		**22**	**1**	**19**	**2**

Table 6: LSA Results, Decision Tree Classifier

Feature Vector	Classifier	Model Number	Complete Data		Incomplete Data	
Baseline:	N/A	M0.0	381	7	353	6
Random			23	0	20	1
LSA	Decision Tree	M1.1	370	18	336	23
	max_depth=15		20	3	19	2
LSA	Decision Tree	M1.2	368	20	**329**	**30**
+Mentions	max_depth=15		18	5	**17**	**4**
LSA	Decision Tree	M1.3	376	12	335	24
+Dependencies	max_depth=15		20	3	20	1
LSA +Mentions	Decision Tree	M1.4	370	18	337	22
+Dependencies	max_depth=15		20	3	20	1
LSA.content	Decision Tree	M1.1b	**377**	**11**	346	13
	max_depth=15		**19**	**4**	19	2
LSA.content	Decision Tree	M1.2b	375	13	334	25
+Mentions	max_depth=15		23	0	19	2
LSA.content	Decision Tree	M1.3b	365	23	345	14
+Dependencies	max_depth=15		22	1	19	2
LSA.content +Mentions	Decision Tree	M1.4b	367	21	342	17
+Dependencies	max_depth=15		23	0	20	1

Table 7: word2vec and GloVe Results, Logistic Regression Classifier

Feature Vector	Classifier	Model Number	Complete Data		Incomplete Data	
Baseline:	N/A	M0.0	381	7	353	6
Random			23	0	20	1
word2vec	Logistic Regression	M2.1	**382**	**6**	354	5
	C=46.4; C=100		**18**	**5**	21	0
word2vec	Logistic Regression	M2.2	383	5	354	5
+Mentions	C=4.6; C=215.4		20	3	21	0
word2vec	Logistic Regression	M2.3	382	6	352	7
+Dependencies	C=10; C=2.2		19	4	21	0
word2vec +Mentions	Logistic Regression	M2.4	386	2	352	7
+Dependencies	C=10; C=2.2		19	4	21	0
GloVe	Logistic Regression	M3.1	384	4	353	6
	C=46.4; C=21.5		20	3	20	1
GloVe	Logistic Regression	M3.2	384	4	354	5
+Mentions	C=464.2; C=100		20	3	20	1
GloVe	Logistic Regression	M3.3	383	5	**354**	**5**
+Dependencies	C=215.4; C=21.5		21	2	**19**	**2**
GloVe +Mentions	Logistic Regression	M3.4	382	6	**354**	**5**
+Dependencies	C=464.2; C=21.5		21	2	**19**	**2**

Table 8: word2Vec and GloVe Results, Decision Tree Classifier

Feature Vector	Classifier	Model Number	Complete Data		Incomplete Data	
Baseline:	N/A	M0.0	381	7	353	6
Random			23	0	20	1
word2Vec	Decision Tree	M2.1	**372**	**16**	346	13
	max_depth=15		**17**	**6**	21	0
word2Vec	Decision Tree	M2.2	364	24	335	24
+Mentions	max_depth=15		23	0	19	2
word2Vec	Decision Tree	M2.3	375	13	344	15
+Dependencies	max_depth=15		19	4	18	3
word2Vec +Mentions	Decision Tree	M2.4	361	27	337	22
+Dependencies	max_depth=15		22	1	19	2
GloVe	Decision Tree	M3.1	377	11	347	12
	max_depth=15		20	3	18	3
GloVe	Decision Tree	M3.2	366	22	**349**	**10**
+Mentions	max_depth=15		21	2	**18**	**3**
GloVe	Decision Tree	M3.3	372	16	348	11
+Dependencies	max_depth=15		20	3	20	1
GloVe +Mentions	Decision Tree	M3.4	365	23	345	14
+Dependencies	max_depth=15		22	1	19	2

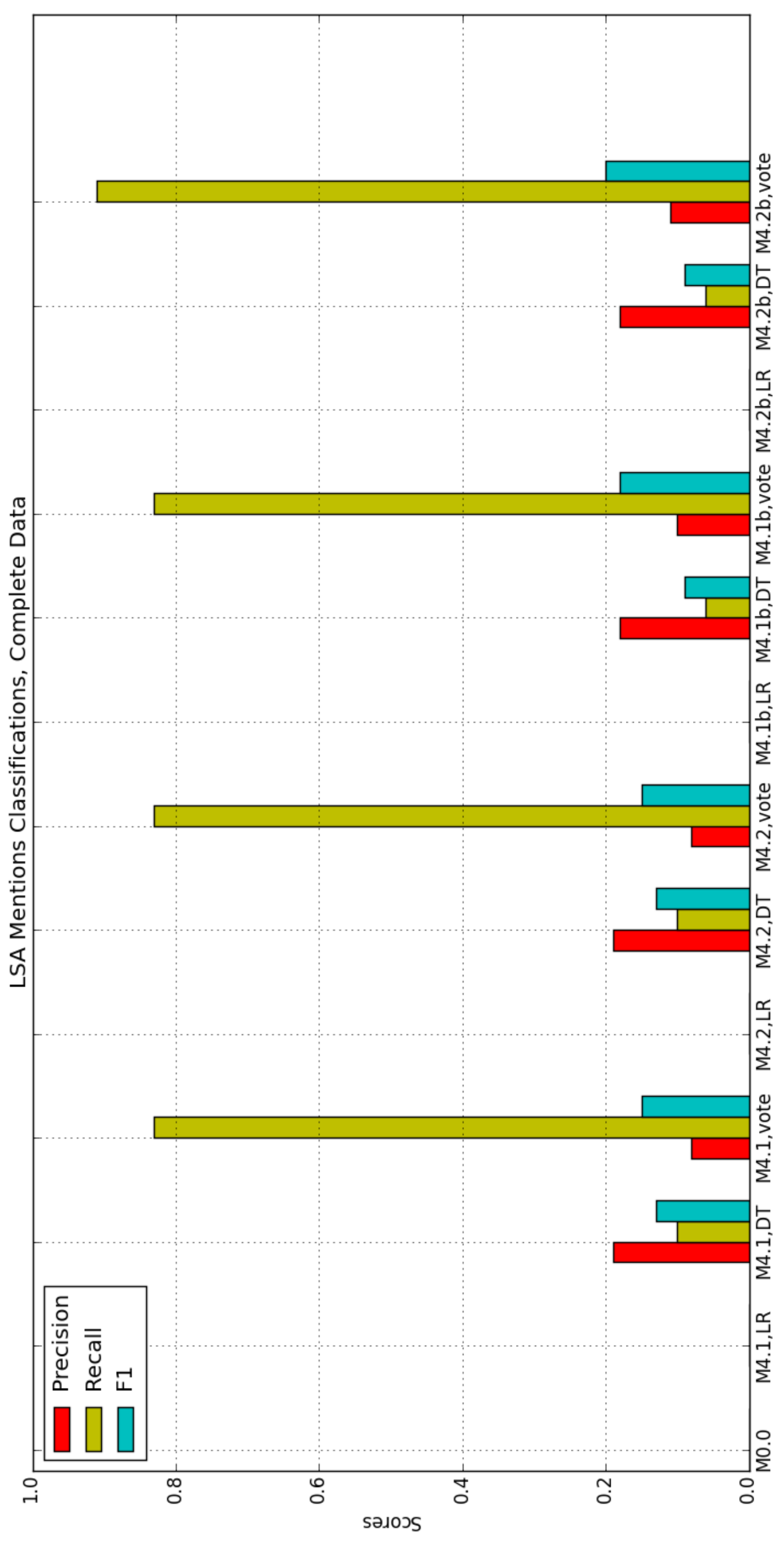

Figure 11: LSA Mentions Classifications, Complete Data

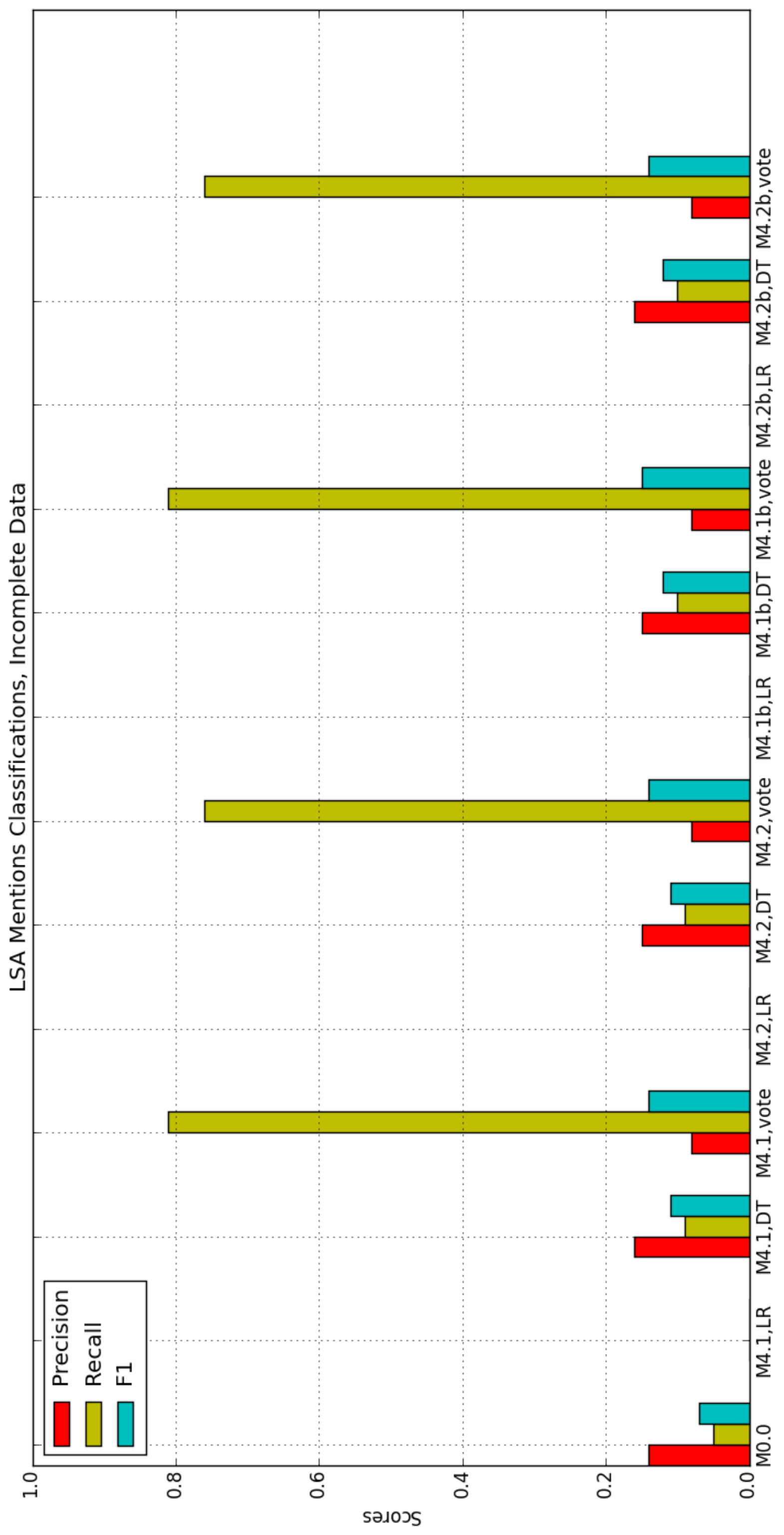

Figure 12: LSA Mentions Classifications, Incomplete Data

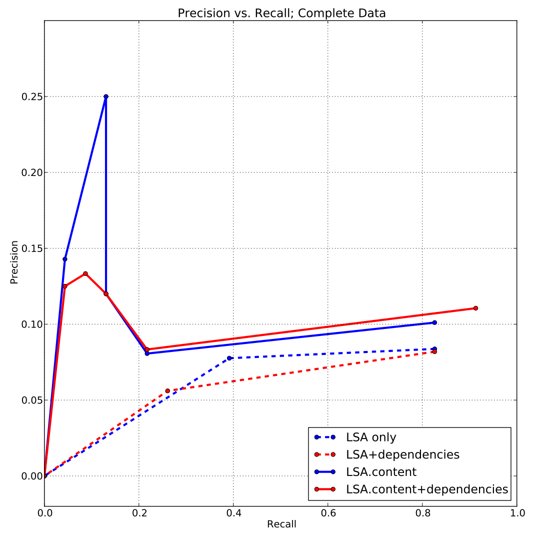

Figure 13: PR curve, Complete Data

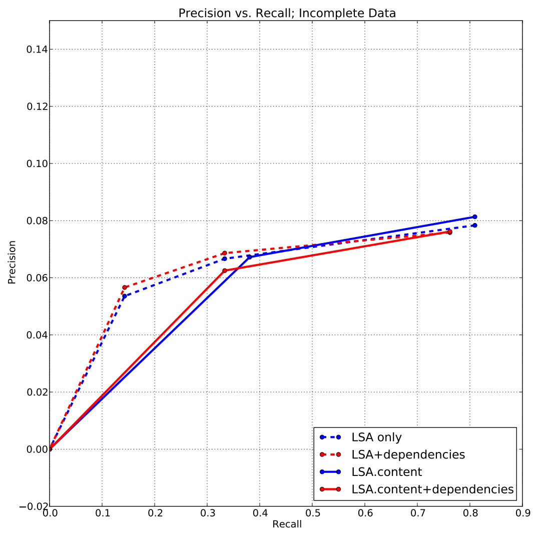

Figure 14: PR curve, Incomplete Data

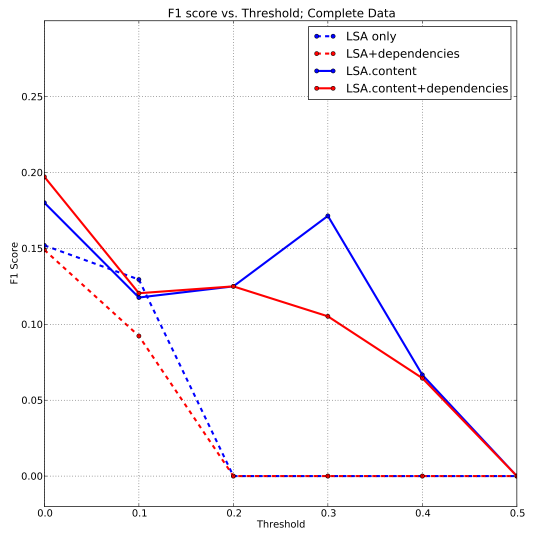

Figure 15: F1 curve, Complete Data

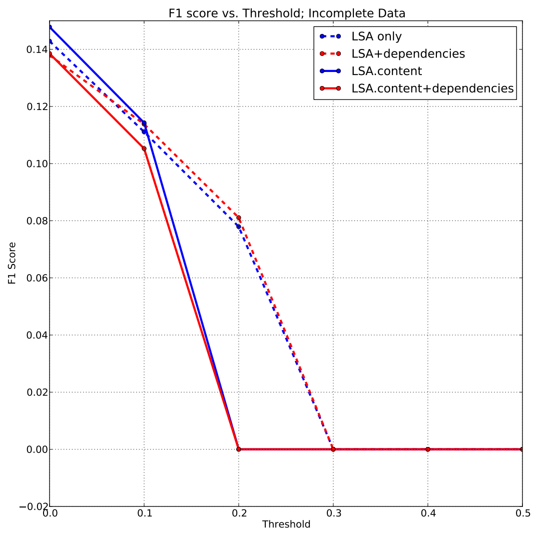

Figure 16: F1 curve, Incomplete Data

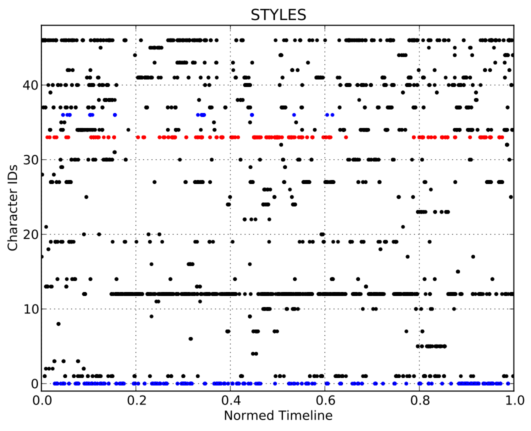

Figure 17: *The Mysterious Affair at Styles*

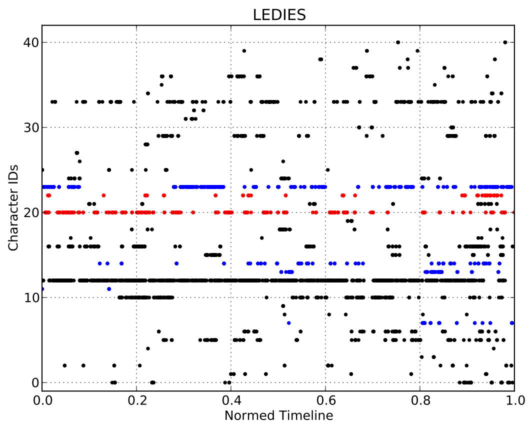

Figure 18: *Lord Edgware Dies*

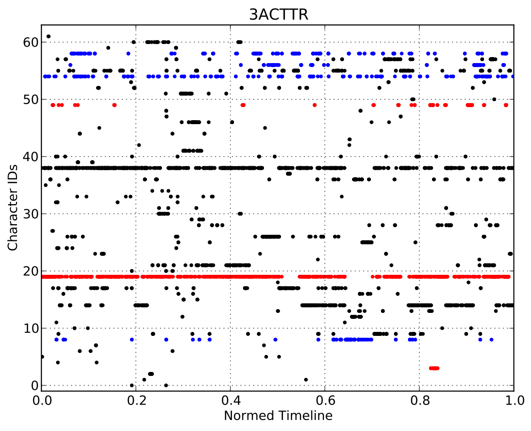

Figure 19: *Three Act Tragedy*

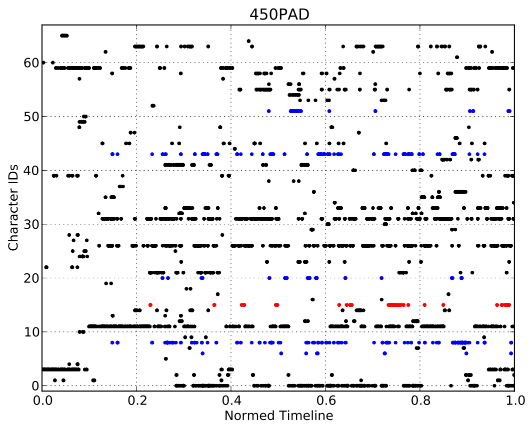

Figure 20: *4:50 from Paddington*

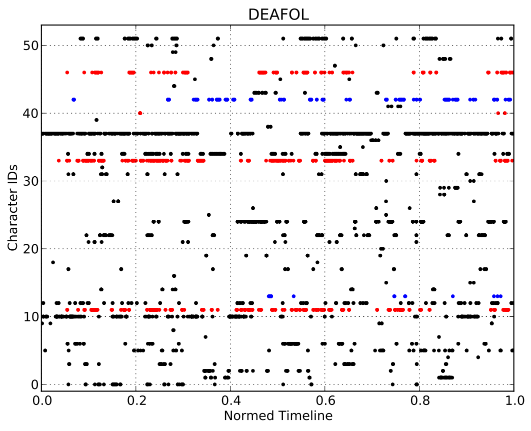

Figure 21: *Dead Man's Folly*

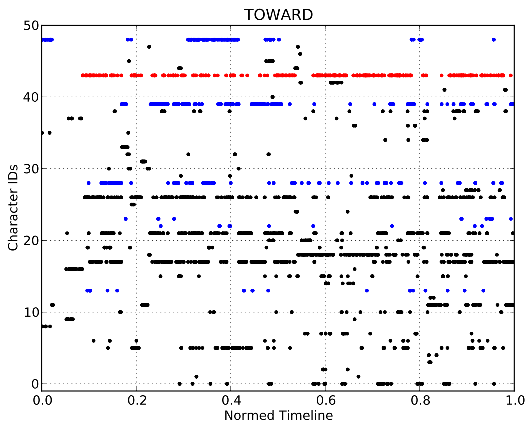

Figure 22: *Towards Zero*