Who's Who?

Learning To Identify a Character

Using Only a Qualitative Description and the Full Text of a Book

Henry Aspegren, Yiqun Hu, Dominiquo Santistevan

6.864 Final Project

Abstract

Generating high-level abstract descriptions of characters from the text of a book is a uniquely challenging problem. It requires substantial inference beyond merely looking at relationships or keywords. The system must be able to identify what makes a character have a certain quality, which is usually not stated explicitly. While many information extraction systems exist, few have attempted this challenging task. We formalize this as a classification problem: given an anonymized character description (characters name stripped out), the name of a character and the entire text of a book, we output a binary decision for whether the description matches the character named. To accomplish this we must determine which parts of a book are relevant to a particular character. Using a collection of 3,932 SparkNotes character descriptions and the full texts of 237 books we develop three models: an SVM using averaged word representations over the whole text, an SVM using averaged word representations coupled with a simple text-extraction protocol and a neural model using two RNNs to learn deep representations for description and extracted text with a feed forward NN. The bag-of-words approaches perform poorly and are unable to abstract away character qualities. The Neural approach shows promise yet its effectiveness is limited by the subtask of identifying relevant text for a character.

1 Introduction

Generating high-level abstract descriptions of characters from the text of a book is a uniquely challenging problem. It requires substantial inference beyond merely looking at character relationships or keywords. The system must be able to identify what makes a character have a certain quality, which is usually not stated explicitly. For instance, in the following character description for Ophelia in *Hamlet*[1],

> Ophelia is a **sweet** and **innocent** young girl, who **obeys** her father and her brother, Laertes. **Dependent on men to tell her how to behave**, she gives in to Polonius's schemes to spy on Hamlet, she gives in to Polonius's schemes to spy on Hamlet. Even in her lapse into **madness** and death, she remains **maidenly**, singing songs about flowers and finally drowning in the river amid the flower garlands she had gathered.

Ophelia's personality was inferred from a series of actions she took and what she said, which are processed by human brains to obtain certain abstract descriptions such as "innocent" or "dependent". This could be a hard task even for human! Therefore, while many information extraction systems exist, few have attempted this challenging task.

1.1 Objective

While the ultimate goal can be as complicated as developing a model that can generate character descriptions with a specified name and full text of the book, we decide to start formularizing this problem with two simpler tasks:

[1]from SparkNotes

1. build a classification model that, with anonymized character description and book text, determines whether this character is actually inside the book
2. build a matching algorithm that, based on related text from the book, selects the best matching character description from a pool of characters

1.2 Related Work

We have not found many research work related to our project. The paper *Extracting Social Networks from Literary Fiction* by Elson, Dames, McKeown[1] also analyzed characters in books, but with a focus on the inter-relationships among characters.

1.3 Project Outline

In section 2, we will explain how we constructed our database and preprocess them for modeling. In section 3, we will describe the mathematical formulation of the problem. In section 4, we will show some of the results of our algorithms and give a breif discussion in the findings. In section 5, we provide some suggestions for future research.

2 Data and Preprocessing

We constructed our database mainly from two sources:

SparkNotes: provides high-level abstract summarization for characters, for example

Gutenburg: provides many free digitized books

We gathered a collection of 3,932 SparkNotes character descriptions and full texts of 237 books.

2.1 Word Embedding

After some preprocessing of the raw text and character descriptions, such as anonymizing character descriptions, we need to convert them into vectorized representations. Instead of using a one-hot vector encoder, we created word embeddings learned from on our entire corpus. To achieve this, we used Gensims *word2vec* module and trained it on all the main texts and character descriptions. This produces a mapping from each word in our text to a 100 dimensional real valued vector. Note that if the given word appeared fewer than ten times in the text, we defaulted to a 100 dimensional zero vector. We note this is an area of potential improvements given that language varies in text on period and region. However, we were not able to train them separately due to the number of books we have in our database.

2.2 Relavant Text Extraction

Since one book can contain many characters, using the same book with all the text can create huge noise]. Therefore, we decide to construct a way to retrieve relevant text for a certian character. This is no trivial task in itself. The subtly of identifying whether a character is talking, being talked about, or simply referenced involves many layers of analysis that are tough to validate. We sidestepped this problem by making a generalization, but we understand that this problem alone could be extremely beneficial for increasing an AIs "understanding" of a book.

For each character in a book, we decided that the 'relevant text' for that character would be the window of text 500 characters on each side of the place where a character was explicitly mentioned by checking their name and a combination of possible names that may be used to reference that character. Now, for each book we had a list of text excerpts where that character's name or nickname was mentioned, which was a theoretically much more specific in describing who a character is in the scope of the full text.

With our text partitioned into several different ways, we now needed to get text from its more raw form, a series of words, to a vector representation. Using our embeddings model, we represented a document by taking the average of all embedding vectors for each word in our document. Whether the document was the full text, character description or relevant text, this approach gave us a 100 dimensional vector that essentially represented a single word for that document.

2.3 Unbalancedness

Since each character only occurs in a single book there is one positive example for that character and 236 negative examples. This left us with an extremely unbalanced set of points, so we decided to randomly sample the negative pairs, and obtain a 50/50 ratio for the data to be input into the model.

3 Approach

3.1 Averaged Word Baseline Model

Our first approach set forth to answer the question of:

> Given text and character description, does the description describe a character in the main text?

This would be a binary problem trained on pairs of vectorized character texts and main texts using the average vector approach mentioned earlier and concatenating the two vectors together. An SVM classification is applied to output a binary reponse, with 1 indicating the character is in the text, and 0 otherwise. The framework of the model is illustrated in Figure 1.

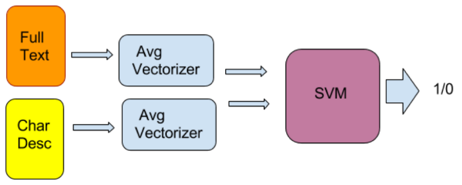

Figure 1: Average Word Baseline Model

3.2 Averaged Word Baseline with Extraction Protocol

The next approach was slightly more complex and this involved the extraction method for relevant text for a character, as explained in section 2.2. We now wanted to describe, given a vector representation of an excerpt of a books main text and description vector, whether this description match the relevant text for the character. In this model, we made an additional step to extract relevant text and calculate the average vector representation for it. The framework of this model is illustraed in Figure 2.

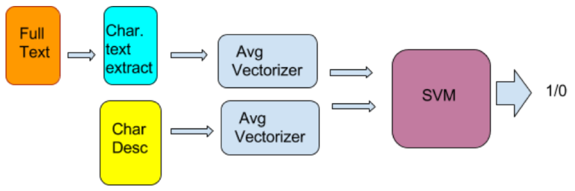

Figure 2: Averaged Word Baseline with Text Extraction

3.3 Neural Net Approach

Neural networks are known to perform well in discriminative models. So the third model we propose is the neural net approach, which takes in the same input as the previous model, but with a more sophiscated structure underneath. The relevant text and description are converted to variable length vectors by mapping each word to a fixed length vector using word embeddings learned from the full dataset. These vectors are feed into two RNNs - one RNN for the text and one RNN for the description each using *tanh* activation functions for the hidden units:

for $* \in \{text, desc\}$:

$$z_i^* = W^*[{w_i^*}^T \quad {h_{i-1}^*}^T]^T + b^*$$
$$h_i^* = \tanh(z_i^*)$$

where z_i is the state at step i, w_i is the averaged embedding for input at step i, h is the activation, and W^* and $b*$ are weights and biases to be learned.

The outputs of the two RNNs are concatenated together to get a fixed length vector representing both the text and the description. This is the input to a feed-forward neural network layer with *rectified* linear activation functions:

$$z^{FF} = W^{FF}[{h_o^{text}}^T \quad {h_o^{desc}}^T]^T + b^{FF}$$
$$f^{FF} = \mathrm{ReLU}(z^{FF})$$
$$z^o = W^o \cdot f^{FF} + b^o$$

Finally we use a softmax layer with two classes - 1 for “yes” and 0 for “no” - to get the probability that the description is of the character. An output label is chosen using the maximum probability class:

$$p_0 := P(\text{desc not in text}|w^{text}, w^{desc}) = \frac{e^{z_0^o}}{\sum_{i\in\{0,1\}} e^{z_i^o}}$$
$$p_1 := P(\text{desc in text}|w^{text}, w^{desc}) = \frac{e^{z_1^o}}{\sum_{i\in\{0,1\}} e^{z_i^o}}$$

The framework of our neural net approach can be seen in Figure 3.

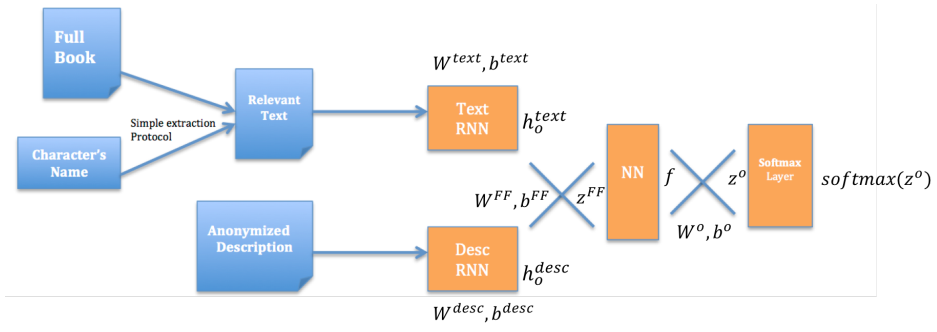

Figure 3: Neural Net Approach

The objective function for training the network is the cross entropy loss:

$$l(p) = \sum_{i\in\{0,1\}} q_i \log p_i$$

where q_i is 1 if the descripted character is actually in the book and 0 otherwise.

We used Tensorflow to learn the weights in the overall network described above and this process can be seen in Figure 4. The entire architecture is trained simultaneously using stochastic gradient descent, for which we feed small batches to the algorithm in terms of different book pairs. Then one layer of softmax is to find the highest probability for the match character.

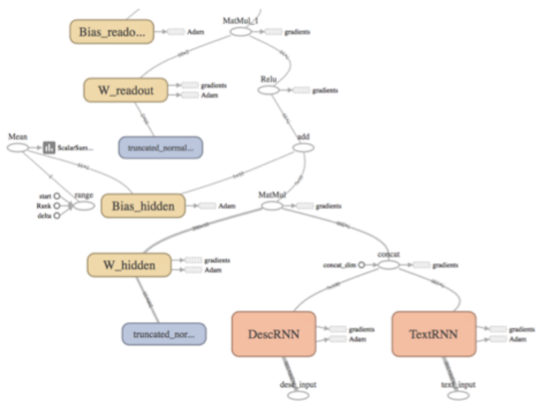

Figure 4: Training Network in TensorBoard

3.4 Matching Algorithm

The matching algorithm differs from the neural net approach in the third feed-forward sub net. Instead of feeding the final states from the two RNN's, we calculate the cosine similarity for each pair of relevant text representation and character description.

4 Results and Discussion

The following table summarizs all three models' performance in terms of precision, recall and overall accuracy:

Model	Set	Precision	Recall	Accuracy
Baseline	Train	0.587	0.352	0.551
	Test	0.552	0.340	0.542
Text Extraction	Train	0.536	0.532	0.536
	Test	0.511	0.528	0.514
Neural Approach	Train	0.783	0.698	0.732
	Test	0.674	0.621	0.665

Table 1: Model Performance

Based on the results we see above, we make some of the following conclusions:

- The baseline models perform less satisfactory as we have high dimensional data, which do not usually work with simple linear SVM
- There is a big increase in model performance using the neural net approach due its capability of sparating highly nonlinear data
- By feeding pairs of books in sequence due to memory constraints, the neural model suffers from overfitiing to specific book pairs; we conjecture a randomly shuffled data would increase the model performance
- Due to unbalancedness in our dataset, we dropped a lot of negative data samples, which results in a worse recall than precision overall
- We think the model performance can be further increased by tuning hyperparameters such as the length of the embedding vector, number of nodes in hidden layers, etc.

5 Future Research

We realize that we had to make a lot of specific choices in many of the intermediate steps, such as how to extract relevant text, etc, which themselves could be an individual NLP research projects. Therefore, there are many potential areas that can be improved, which can be future research directions motivated by this project, for instance:

- Allow different embeddings for same words across diffrent periods and regions
- Build more sophiscated extraction scheme for related text
- Optimize how data gets feed into the model for more efficient calculation and higher accuracy
- Build a generative model that can produce simple character descriptions
- Incorporate with language models, to produce readable sentences as character descriptions

6 Contributions

Dominiquo Santistevan:

Most of my work was done in creating the classes that would become the foundation for transforming text to vectors. This included extracting raw text into something more interpretable with the 'Book' class. This class was just an easier representation for retrieving any raw text for a book. I also wrote the code for the "SimpleVectorizer" and "RelevantTextVectorizer" which took the text from the book class and produced the vectors that would be used in our model. Another contribution of mine was creating the "BookPairs" functions which took two books and created the batch of vectors for matching descriptions and characters. This involved randomization and padding of the vectors such that we were consistent with format for feeding them into our neural model. We all contributed towards collecting the data and cleaning what we could by hand which actually took a decent amount of time to complete.

Henry Aspegren:

I wrote much of the code used in the project. I authored all the scripts in the helper_utils directory to scrape the data from sparknotes, clean the descriptions, partition into train and test examples and overall build a workable datasets. Using the foundation classes built by Dominiquo I implemented the Neural Model using Tensorflow and the Basic Model using scikit learn. I wrote the code needed to feed the large quantities of data into the system quickly and efficiently. Specifically I wrote generators to feed the data into the tensor flow computation graph that lazily vectorize the books without overloading the system resources. This was the most challenging technical aspect of the project for me.

Yiqun Hu:

We first distributed the work of manually collecting data and some preprocessing evenly among the three of us. I then worked together with Henry in establishing the neural net approach, constructing formularized the mathematical representations of our neural model as seen in the report. I finished setting up the framework for the matching algorithm, and started working on the code, which is built upon Henry's neural model with different feed_dict and loss function implementation for the different objective. Then I analyzed some of the output from the three model and sought for feasible explanations of the results that we observe, incorporated with our main storyline and integrated everything into a research poster.

References

[1] David K. Elson, Nicholas Dames, Kathleen R. McKeown Extracting Social Networks from Literary Fiction *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL 2010).*