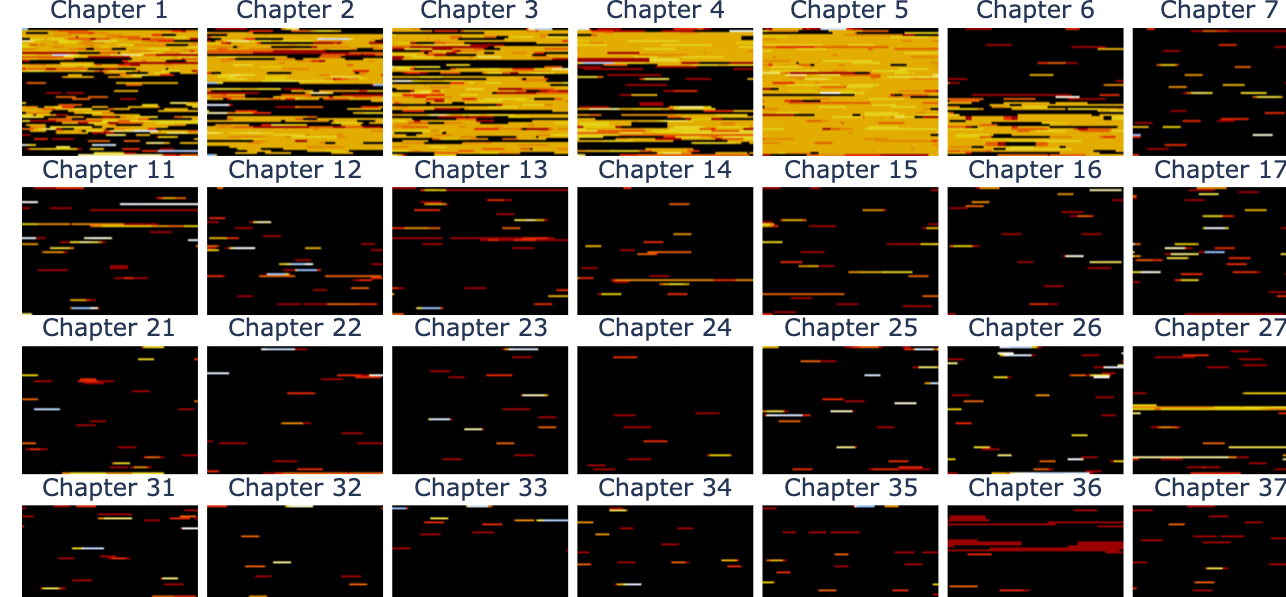
# Machine-aided detection of sources in Jinpingmei: Style, text classification, and the complex textual antecedents of a late sixteenth century Chinese novel.

### Paul Vierthaler[![orcid](data:text/html; charset=UTF-8;base64,)](https://orcid.org/0000-0002-2135-9499) Princeton University

[cc-by](https://creativecommons.org/licenses/by/4.0/) ©Paul Vierthaler. Published by De Gruyter in cooperation with the University of Luxembourg Centre for Contemporary and Digital History. This is an Open Access article distributed under the terms of the [Creative Commons Attribution License CC-BY](https://creativecommons.org/licenses/by/4.0/)

from IPython.display import Image, display  
  
display(Image("./media/cover.png"))



textual history, source detection, machine learning, digital humanities, ming dynasty fiction, chinese fiction

The infamous late sixteenth century novel *Jinpingmei* (*Plum in the Golden Vase*) is renowned as one of the greatest novels of the Chinese tradition and is famed for its extensive intertextuality and complex appropriation of earlier cultural material. The pseudonymous author of *Jinpingmei* spins a masterful and highly pornographic story out of a brief episode from an earlier novel the *Water Margin* and peppers the adapted narrative with intrusions of material from a dizzying array of external works with seemingly no regard for the boundaries of genre. This complex intertextual landscape has attracted the attention of scholars in the hundreds of years since the novel first began to circulate, but now computational methods enable us to systematically reconstruct the novel’s textual origins. In this paper, I propose a methodology to broaden our understanding of this intertextuality using machine-learning based text classifiers to identify the likely textual origin of each instance of text reuse identified within a digital corpus, using the intuition that materials endogenous to *Jinpingmei* will be more stylistically similar to the rest of the novel than exogenous materials (and conversely, that external materials will more greatly resemble the external work than *Jinpingmei*). In this piece, I demonstrate the value and pitfalls of machine-aided detection of source material when working with complex textual artifacts like *Jinpingmei* and comment on the broader applicability of these methods to other literary works from China.

## Introduction

The earliest extant edition of the late Ming novel *Jinpingmei* (*Plum in the Golden Vase* 金瓶梅), known as the *cihua* 詞話 or Wanli 萬曆 edition (named after the Wanli reign, 1570--1620), hereafter *Jinpingmei*, opens not with original material but with a reference to an older ci lyric poem:

The hero grips his “Hook of Wu.” Eager to cut off ten thousand heads. How is it that a heart forged out of iron and stone, can yet be melted by a flower? Just take a look at Hsiang Yü and Liu Pang: Both cases are equally distressing. They only had to meet with Yü-chi and Lady Ch’i for all their valor to come to naught” ((Roy, 1997), 12). 詞曰：「丈夫只手把吳鉤，欲斬萬人頭。如何鐵石打成心性，卻為花柔。請看項籍並劉季，一似使人愁；只因撞著虞姬戚氏，豪傑都休。」

Widely viewed as a masterpiece and one of the most important and innovative works in the Chinese literary tradition, the anonymously written *Jinpingmei* was unique, experimental, and extremely controversial. The novel touches on myriad social, political, religious, and moral issues through the lens of a pornographic story and is viewed as an important allegory lampooning late Ming society. Yet despite the novel’s unique place in Chinese literary history, this very first passage, and significant portions of the novel as a whole, is not original. As Patrick Hanan points out, this same lyric poem (with only very minor differences) can be found in the late fifteenth century Hong Pian’s 洪楩 *Qingpingshan huaben* 清平山堂話本, a collection of short stories referenced throughout *Jinpingmei* ((Hanan, 1963), 25). Critically for Hanan’s argument, it is not just the poem that is copied; the novel reproduces the explanation for the poem that immediately follows nearly verbatim:

The subject of this lyric is the words passion and beauty, two concepts that are related to each other as substance is to function. Thus, when beauty bedazzles the eye, passion is born in the heart. Passion and beauty evoke from ancient times until the present day, gentlemen of moral cultivation ought never to forget. As two men of the Chin dynasty once said, “It is people just like ourselves who are most affected by passion ((Roy, 1997), 12). 此一只詞兒上詩詞各一首，單說著情色二字，乃一體一用也。故色絢於目，情感於心，情色相生，心目相視。雖亙古及迄今，仁人君子，弗合能忘之。晉人云：「情之所鍾，正在我輩。」

Patrick Hanan uses this as evidence that the author of *Jinpingmei* was almost certainly copying the *Wenjing yuanyang hui* 刎頸鴛鴦會 short story within *Qingpingshan*, though this story in turn owes parts of its composition to even older works; buried within this short section of text are references to older materials from the Song, Yuan, and Ming dynasties.

Opening the novel with recycled material immediately primes the reader to expect such moves throughout the novel. The *cihua* edition immediately pivots from this relatively short instance of intertextuality to a more widely recognized, and much longer, adaptation of material from the novel *Water Margin* (*Shuihu zhuan* 水滸傳). After the significant opening interlude, the novel constantly returns to earlier material. The complex intertextual references within *Jinpingmei* are so dense that well over five percent of the novel derives from earlier sources.

Contemporary readers would have been steeped in the quoted works, and these references would have likely elicited a variety of responses dependent on knowledge of the earlier works. Yet even they clearly found this material difficult, and very few modern readers have enough insight into the sources to fully appreciate the effect they have on the novel. Many details of this intertextuality remain elusive, even in terms of the most clear cases. For example, the specific version of the *Water Margin* the author used is some matter of debate, and Hanan argues that the source edition is distinct from the many currently extant versions. These problems are not unique to modern readers either, as late imperial readers clearly felt the *Water Margin’s* influence on *Jinpingmei* controversial. Later editions of the *Jinpingmei*, such as the much more popular Chongzhen-era (1628--1644) edition, significantly reframe the relationship between the two novels by excising and editing large amounts of the shared text.

Following the permutations of textual adaptation within the novel offers a deeper understanding of the work, but it is an arduous process that requires an extremely in-depth knowledge of the works circulating when *Jinpingmei* was written. *Jinpingmei’s* role as an exemplar in the art of textual appropriation makes it a perfect test case for developing digital methods for automatic detection of intertextuality and disambiguation of sources. Patrick Hanan’s 1963 article “Sources of the Chin P’ing Mei” is a masterclass in textual scholarship, in which he identifies many of the works directly copied within the novel. But digital methods and materials open the door to such detailed scholarship much more widely. Critically, they allow me to generate a generalizable approach that works beyond the case of *Jinpingmei* that may be valuable in studying all manner of literary and historical materials.

In prior work I developed and adapted a method that makes identifying instances of intertextuality at corpus scale feasible ((Vierthaler and Gelein)). However, this approach left the issue of quotational direction unanswered, providing no clarity in terms of which text was quoting which. In this article, I discuss experiments I have been conducting that leverage stylistic signals found within digital corpora and intertextual material to computationally identify the textual antecedents deployed so readily within *Jinpingmei*. This process is critical for revealing more about the nature of *Jinpingmei* as a heteroglossic, intertextual work, but it also leads to the creation of tools useful for studying textual sources within Chinese materials writ large. Specifically, I will introduce a workflow in which I train various machine-learning models that operate on stylistic markers (primarily the relative use of certain words/characters) to evaluate the likely direction of textual sharing between *Jinpingmei* and related works. This process rests on the general assumption that a quote is more similar to the text it originates from than to the text is has been inserted into. This is primarily useful for cases where the quotational direction cannot be ascertained through purely chronological means, developing an approach to identifying quotations that is not possible absent computational analysis.

The workflow and tools I show here have the potential to expand the study of source materials in Chinese corpora far beyond the use cases found within *Jinpingmei*. The work in this article is also leading into a future study of the stylistic nature of *Jinpingmei* that illustrates how intertextuality and reliance on materials of multiple different genres influences the style of the novel.

## Acquiring and prepping the corpus

The quality of any large-scale textual analysis depends on the digital corpus that undergirds the research. It is only thanks to work by people who have digitized a large portion of imperial Chinese writing that it is now possible to leverage computational analysis to understand the works that influence any given text on a macro scale. I base the analysis in this paper on digitized texts derived from a number of online repositories: the Kanseki repository ((Wittern, n.d.)), the Daizhige repository ((Daizhige, n.d.)), Chinese Wikisource ((Chinese Wikisource, n.d.)), as well as a number of other open-source collections including Project Gutenberg ((Project Gutenberg, n.d.)) and *Kaifang wenxue* ((Open Lit, n.d.)).

I first clean the texts in order to facilitate the identification of shared textual materials across different digital corpora. This involves removing materials not likely present in the original works, from punctuation to errant HTML. I also normalize the Chinese character sets across documents by transforming the texts into simplified characters. This is necessary because the corpus comprises materials in both traditional and simplified character sets. Ideally, I would transform the simplified documents into traditional ones. However, because a significant number of traditional characters were simplified from multiple characters into a single simplified graph, it is difficult to automatically transform a simplified text into a traditional one. Going in the other direction loses some information but ensures a uniform, determinative character set. I removed duplicate materials in cases where a particular edition of a text exists as multiple *digital* copies across this corpus, but I do retain different editions of texts if they represent a unique printed piece. For most of the analysis, I also ensured that I have a rough date of composition for every text. In cases where it is not clear what dynasty a work is from, I still conduct the intertextual search across the material but leave it out of the bulk of analysis. I also excluded all editions of the *Jinpingmei* from the comparative corpus to simplify the analysis.

Given that my primary focus is on identifying the sources of *Jinpingmei*, simply knowing if a text that shares material with *Jinpingmei* was written before the novel is enough to establish that the materials is not original to *Jinpingmei*, though not necessarily to identify its ultimate origin. As such, dating the works in the corpus becomes critical. This is complicated by the fact that the exact date of writing of *Jinpingmei* is not clear: it was almost certainly written in the second half of the sixteenth century and finished no later than 1606, when we have evidence that full copies of manuscript versions of the text were circulating. Shen Defu discusses the existence of several full copies of the manuscript in the *Wanli yehuo bian* 萬曆野獲編, referring to an incident that occurred in 1606 in which he discussed the novel with Yuan Hongdao. Shen references a complete copy owned by Xu Wenzhen (1503-1583), pushing likely completion back to at least the late sixteenth century ((Xu, 2011), 72). In including materials written roughly before, concurrently to, and after *Jinpingmei*, I can get a deep view of its influences and the impact it has on later materials. In the end, the analysis corpus I use contains 20,637 texts totaling around 1.58 billion characters. The details of the subset of works that share text with *Jinpingmei* are described in the table below:

|  |  |  |
| --- | --- | --- |
| Temporal Relation to *JPM* | Number of Texts | Total Characters |
| Predates | 1,560 | 326,606,139 |
| Contemporary/Unknown | 1,406 | 292,256,678 |
| Postdates | 1,599 | 492,891,996 |
| Totals | 4,565 | 1,111,754,813 |

The quality of the corpus heavily influences the results of any corpus-based research. Transcription errors could influence the results, for example. When comparing two different editions of the same work, it is important to remember that modern digitizers might have introduced many of the differences, so the digital editions may not completely reflect the original works. However, given the breadth of analysis, as long as the transcription errors are not systematically biased, they should not have a significant influence on the results at hand.

It is also the case that the scope of analysis is limited by works represented within the corpus: I can often trace the likely source of a given chunk of text but I can only go as far back as the corpus allows, and it is entirely possible that the corpus does not contain the ultimate source. Still, the material represented by the corpus here offers an excellent starting place for comprehensively studying *Jinpingmei's* source material.

## Identifying Intertexuality

The first step in computationally identifying the sources of *Jinpingmei* is to compare the novel against all materials within the corpus. While it is relatively simple to identify sections of text which are direct quotations, often the original is not preserved exactly; the author of *Jinpingmei* clearly felt at liberty to edit materials he incorporated into the novel. As such, I use an algorithm based on bioinformatic’s Basic Local Alignment Search Tool (BLAST) to ensure that as long as there is a set amount of similarity between *Jinpingmei* and the comparison text, I identify all shared material. This method, and the software that implements it, is described in Vierthaler and Gelein, 2019.

There are a variety of other methods one could use to detect intertextuality found in the *Jinpingmei* (or any other work for that matter), including approaches developed by Donald Sturgeon (as described in ((Sturgeon, 2018))) or by Jeffery Tharsen and Clovis Gladstone (as described in ((Tharsen & Gladstone, 2020))). The advantage the BLAST-based method has is that it is optimized to work in a Python-based workflow run locally on large corpora. The flexibility this gives me allows me to rapidly prototype new output formats and facilitate downstream analysis.

When I perform the intertextuality analysis, I look for all sequences of at least ten sequential characters in the *Jinpingmei* that are at least 80 percent the same as sequences of text found in works in the target corpus. I calculate this 80 percent similarity using Levenshtein distance, which essentially measures the edit distance between two strings by calculating how many edits change one string into another. So for a 10-character string, 2 edits are allowable. For a 100-character string, 20 are allowable, and so on. I also align and clean the results for easier display and interpretability, which includes dropping mismatched trailing characters (so the alignment dataset contains many identical 8-character strings).

While the exact parameters I use for searching are somewhat arbitrary, I arrived at these in an attempt to find a balance between noise and comprehensive results (that is to say, I balance precision and recall). The shorter and less similar a sequence I allow, the more likely I will find random, rather than meaningful, instances of reuse. Longer and stricter parameters generate less noisy results, but often miss interesting cases of reuse. A search using these 10 character, 80 percent parameters identifies approximately 537,000 cases of textual reuse in the corpus, involving 13,300 unique quotes across 4,565 unique texts. The results are broken down depending on their temporal relationship with *Jinpingmei* in the following table:

|  |  |  |  |
| --- | --- | --- | --- |
| Temporal Relation to *JPM* | Number of quotes | Unique quotes | Total characters in unique quotes |
| Predates | 49,004 | 3,228 | 82,760 |
| Contemporary/Unknown | 99,054 | 3,658 | 37,157 |
| Postdates | 389,364 | 10,792 | 742,677 |

Some of these instances are short and extremely common; some ten characters sequences appear dozens of times per text across hundreds of texts. Others are long and uncommon, spanning hundreds of characters and shared only with one other text in the corpus. One notable phenomenon is that quotes linked to unkown/contemporary are more common than those that predate *Jinpingmei*, but they also tend to be much shorter in length. Note that this approach does not capture extensive paraphrasing or allusions. This would be very useful to capture, but it is much more complex and computationally intensive.

Following the identification of these instances of textual reuse, I sort the quotes into multiple categories: quotes that represents structural language inherent to a given genre, quotes from works that predate *Jinpingmei*, quotes from works roughly contemporary with *Jinpingmei*, quotes from works that postdate *Jinpingmei*, and quotes from works of unknown date. This basic categorization schema then lets me delve into the influences on, and of, *Jinpingmei*. To do so I am using tools derived from stylometric analysis and machine-learning to develop text classification models that establish the likely origin of textual material. I first establish the efficacy of such models and their potential pitfalls by testing them on a variety of known cases: the *Qingpingshantang huaben*, Li Kaixian’s *Cinue* 詞謔, and the *Water Margin*, before then attempting to ascertain the likely origin of contemporary and unknown materials.

## The nature of intertextuality in *Jinpingmei*

Endeavoring to computationally identify meaningful text reuse in *Jinpingmei* necessitates a careful look at what types of language are generally repeated throughout Chinese literature. While the raw results of the intertextuality analysis are comprehensive, at least to the extent that they account for all instances of reuse within the corpus at hand, they are not particularly illuminating in and of themselves. The first major concern has to do with the sensitivity of the algorithm to stock or structural phrases. The single most commonly repeated phrase in the results is a repetition detected over 480,000 times and is some variation or another of “to learn what happens next, listen to the explanation in the following chapter.” This shows the exponential nature of this type of analysis, because the phrase appears nearly 100 times in *Jinpingmei*, and when compared with another text in which it appears 100 times, the algorithm generates 10,000 results. Fortunately, these oft-repeated phrases are relatively easy to identify and filter out as structural components of text not meaningful in terms of textual influence (in doing so the size of the corpus reduces slightly to 4,549 texts).

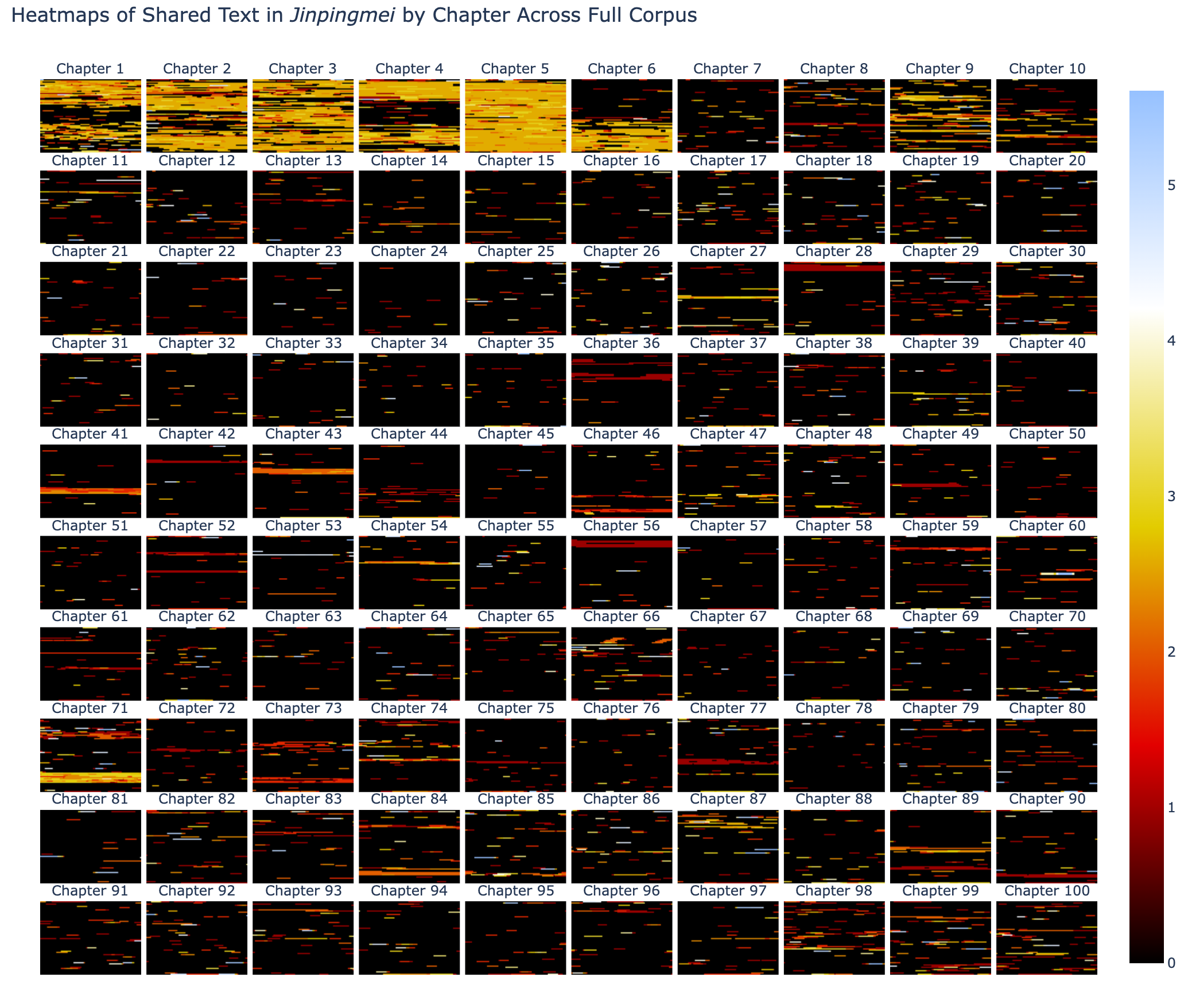
Other phrases also attract prominent attention in view of this study. “Like this and like that 如此如此這般這般” appears thirteen times within *Jinpingmei* and generates over 3,500 results. Some phrases are rarer, appearing only a single time in *Jinpingmei* but appear widely across the corpus. These include “to dance about 手之舞之足之蹈之” and numerous references to various official titles. There are also patterns common in *Jinpingmei* that don’t bear significant repeating across the corpus, usually combinations of the names of certain people or the rather infamous “If Ximen Qing had not heard that would be the end of it, but he heard 西門慶不聽便罷聽了.”

The extent to which quotes are layered is also a complicating factor, as is evident in the opening anecdote of this article. Detecting which of the multiple texts in which a quote appears is the actual source can be difficult. To a certain extent, it is unknowable without commentary directly from the author. It is also difficult to know how we might computationally identify the ultimate origin of the material, which should be understood in contrast with the proximal origin of the material for the author of *Jinpingmei* (as these are not necessarily the same thing). The opening text from the *Qingpingshantang huaben* appears in multiple other places, but like Hanan, we can assume that the longest of the overlapping quotes is the likely progenitor. I will return to this later in the article when the implications are clearer.

I can easily rule out texts that appear after the early circulation of the *Jinpingmei* as possible sources. However, given that I only have a general sense of when in the latter part of the sixteenth century *Jinpingmei* was written, it is more difficult to assess contemporary texts (the issue of unclear composition date is not unique to *Jinpingmei*, so the methods I discuss here will be useful for a variety of other cases). This can lead to cases where the textual overlap in the results does not make intuitive sense as a layered quotation that I can attribute to earlier material. I can assume that the corpus contains at least some cases of quotations that overlap with, but are not completely encompassed by, distinct quotations from earlier periods. In these cases, it may be a later work quoting from *Jinpingmei*. This phenomenon is very clear in texts that clearly quote the *Jinpingmei*, such as Ling Mengchu or Feng Menglong’s various short story collections.

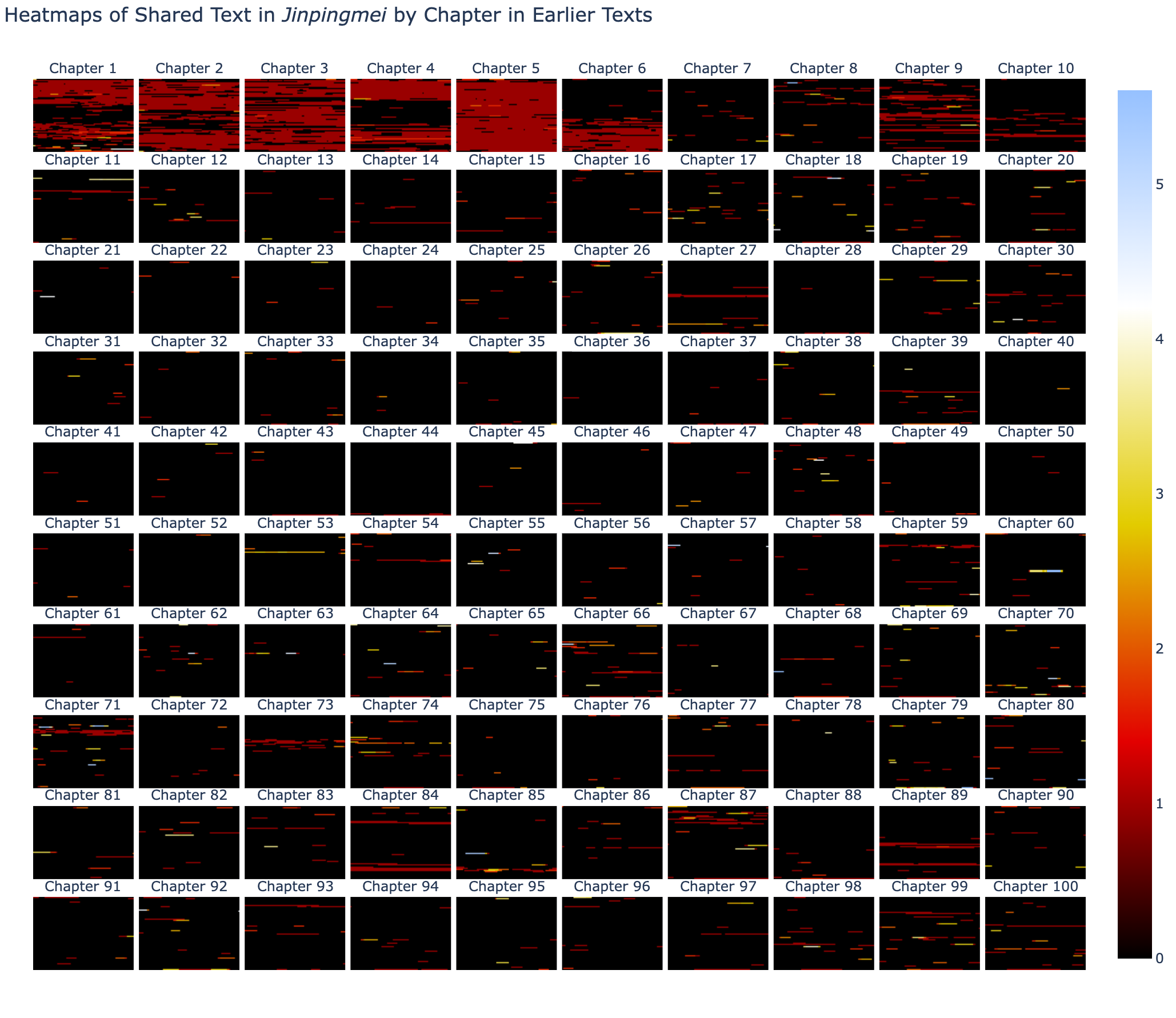
The overall picture of intertextual sharing, including earlier, contemporary, and later materials is as appears in Figure 1 (Discounting only later versions of *Jinpingmei* itself, which largely recapitulate the novel.).

from IPython.display import Image, display  
  
metadata={  
 "jdh":{  
 "module":"object",  
 "object":{  
 "type":"image",  
 "source":[  
 "Figure 1: Heatmaps of Shared Text in Jinpingmei by Chapter Across Full Corpus.",  
 "Shows total number of other texts a given substring appears in. For visual clarity limited to 100 and then the log of the value + 1 is taken."  
 ]  
   
 }  
 }  
}  
  
display(Image("media/heatmapsfull.png", width=1000), metadata=metadata)  
  
# code to generate figures 1 to 3 can be found in the script folder



The above heatmaps illustrate the total number of other texts in the corpus a given substring appears in. For visual clarity I am showing the logarithm of the raw results, because the most common phrases appear so often as to render the rarer results invisible. Substrings in blue appear in 100 or more other texts, substrings in black do not appear elsewhere in the corpus, and substrings in orange appear in 10 to 20 other works. Clearly, the vast majority of intertextuality within the novel appears in the first six chapters of *Jinpingmei*, but many more cases are spread across the entire piece. In order to focus on works that influence the novel’s contents, however, it is necessary to separate out these materials according to date. An exeedingly large number of the intertextual moments evident in this figure are from later works quoting either *Jinpingmei* or shared earlier materials. As such, most of the results originating from earlier or contemporary works are obscured. Figure 2 shows only materials that I can definitively source to works that predate *Jinpingmei*, which gives a clearer sense of the distribution of earlier material.

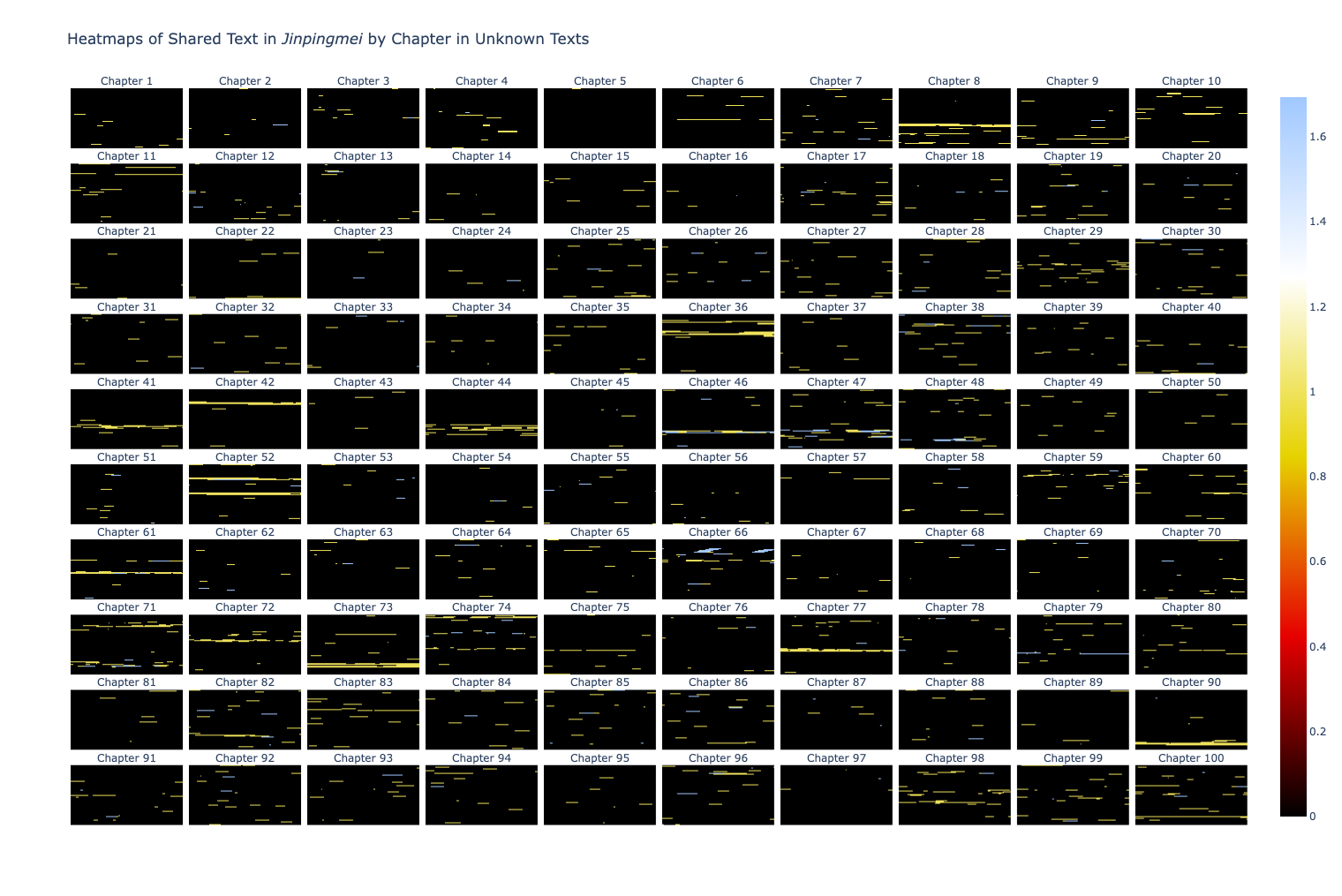
from IPython.display import Image, display  
  
metadata={  
 "jdh":{  
 "module":"object",  
 "object":{  
 "type":"image",  
 "source":[  
 "figure 2: Heatmaps of Shared Text in Jinpingmei by Chapter in Earlier Texts"  
 ]  
   
 }  
 }  
}  
  
display(Image("media/heatmapsearlier.png", width=1000), metadata=metadata)



The earliest part of the novel is still the densest site of intertextuality given the extensive reliance on the *Water Margin*, but there are sections throughout that show significant overlap with earlier material. Chapter 9 in particular is extensively connected with earlier materials. The materials represented in Figure 2 are unequivocally references made within *Jinpingmei* to earlier works.

There are many sections of text whose origins I cannot account for so simply. The numerous sections within the novel that come from roughly contemporary works, and which are difficult to automatically establish as possible source materials, are shown in Figure 3 (I refer to any quotation as "unknown" in the figures throughout this article whenever I do not have external evidence for the direction of quotation).

from IPython.display import Image, display  
  
metadata={  
 "jdh":{  
 "module":"object",  
 "object":{  
 "type":"image",  
 "source":[  
 "figure 3: Heatmaps of Shared Text in Jinpingmei by Chapter in Unknown Texts"  
 ]  
   
 }  
 }  
}  
  
display(Image("media/heatmapsunknown.png", width=1000), metadata=metadata)



References that can only be sourced to unknown materials are abundant but more intermittent than those from earlier texts and computational analysis represents fascinating possibilities for understanding their likely origins. This lower incidence of reference likely derives from teh very high rate of refence to known earlier works like the *Water Margin*, which dominate the intertextual space.

## Deriving the origin of source material

The first step I take in identifying the likely origin of source material in *Jinpingmei* is to filter the intertextual results into several categories. As a preliminary step, I remove the “structural” matches. These are matches like “to see what happens next read the next chapter” that derive from the generic nature of the work rather than its unique style. While these do constitute reuse, these matches do not tell us much beyond the fact that Jinpingmei is a novel that follows usual conventions. I can easily identify this structural material simply by looking for extreme spikes in intertexuality falling around chapter borders.

I then filter the texts by known date of composition, removing materials that were written after *Jinpingmei* began circulating in the late 16th century/early 17th century. I divide the remaining material into two further groups. First is material that predates the novel, which I use to test the machine learning models’ ability to reliably tease out source material. Finally, I have the material that was roughly contemporary with the novel, which form the body of materials that necessitate this kind of investigation (because available chronological evidence is not enough to ascertain the potential textual relationship).

## Textual Representation

In order to create text classifiers, I need to transform natural language textual material into a numerical representation so I can process it through a computer program. There are many ways to do this, and there have been extremely rapid developments in natural language processing in recent years that allow for some very sophisticated representations. Transformer-based models have come to dominate this space as of early 2023 when I conducted these experiments. However, my philosophical approach to developing algorithms is to use the simplest methods possible that provide reliable results. As such, I turn to “bag-of-words” representations of texts (technically bag of *n*-grams) ((Underwood, 2013)). Each text fragment I study is represented as a vector, or list of numbers, whose individual dimensions consist of simple n-gram frequencies. I use the *n*-grams that appear most frequently across the entire corpus to represent the works. I use the implementation of the TfidfVectorizor found in the sci-kit learn Python library ((Sklearn.Feature\_extraction.Text.TfidfVectorizer, n.d.)). This takes care of the vectorization for me while also normalizing the vectors in a way that facilitates the process of training machine learning models. This transforms the textual differences among documents into spatial differences that a variety of algorithms can easily measure and compare.

Once texts have been vectorized, I can use them to train text-classification algorithms to predict the likely point of origin of a particular quote. There are multiple criticisms that rightly emerge from such an approach. Fundamentally, simply looking at *n*-gram frequencies disregards syntactic information and elides the complex nature of words in Chinese. Yet for the downstream task of developing models to identify a quote’s likely textual origin, I will show that such a basic representation performs well.

## Building the text classifiers

The approach I take to predicting the likely source of a quote (which I will refer to as a substring here, as a smaller section of the entire string the represents the whole document) rests on the intuition that the quote substring that represents a given instance of intertextuality is going to be more stylistically similar to the text it originally came from than the text that appropriates it. As such, I randomly extract an equal number of substrings from each document while ensuring that I do not grab text that is shared between the two documents to train my models (this is to say that I ensure that I do not use the shared quotes, which I want the model to predict the origins of, when training the models). Because the quotes I am comparing are generally between 10 and 250 or so characters long, I ensure that the substrings I randomly extract fall at some random length between these two extremes (though the results do contain notably longer quotations as well).

# For easy changes for anyone who would like to reproduce my work, I set the analysis parameters here:  
  
# Please note that the code that actually produces the figures has been commented out  
  
# number of sections to use when training models  
n\_sections = 5000  
  
# length range of sections   
length\_range = (10, 250)  
  
# number of features to analyze  
max\_features = 100  
  
# use idf  
use\_idf = False  
  
# ngram range  
ngram\_range = (1, 1)  
  
# verbose will print out certain results across the notebook  
verbose = False

import random  
  
# set a random seed so the results of running the code are determinative. feel free to change this  
# to see how the performance of the models varies  
random\_seed = 790183  
random.seed(random\_seed)  
  
def get\_random\_sections(text, n\_random\_sections, random\_length\_range):  
 """  
 function to extract n random sections text of a random length from an text.   
  
 First I iteratively create a list of sequences n length chosen at random between two numbers  
 until I've chosen ranges that cover the entire length of the text. Then I randomly sample these  
 sections and return the associated text.  
 """  
  
 # select a starting section  
 sections = [[0,random.randint(\*random\_length\_range)]]  
  
 # As long as the end of the last section is less than the length of the text, keep generated  
 # sections of random length  
 text\_length = len(text)   
 while sections[-1][1] < text\_length:  
 start\_point = sections[-1][1]  
 end\_point = start\_point + random.randint(\*random\_length\_range)  
  
 if end\_point > text\_length:  
 end\_point = text\_length  
 sections.append([start\_point, end\_point])  
   
 # if the last section of text is less than the minimum random length then dispose of it  
 if sections[-1][1] - sections[-1][0] < random\_length\_range[0]:  
 sections = sections[:-1]  
   
 # select a total of n random sections form the list  
 if n\_random\_sections < len(sections):  
 sections = random.sample(sections, n\_random\_sections)  
  
 # return the corresponding text selections  
 return [text[section\_range[0]:section\_range[1]] for section\_range in sections]  
  
  
def get\_alignment\_data(alignment\_file):  
 '''  
 helper function to load in alignment data returned as a nested list  
  
 The alignment file contains 8 columns of data.  
  
 t1\_id: The first text (for jpm\_alignment.tsv this is always 25272, the JPM corpus id)  
 t2\_id: The second text  
 quote\_length: length of shared material  
 quote\_similarity: percent similarity  
 t1\_start\_index: starting index of the quote within the first text  
 t2\_start\_index: starting index of the quote within the second text  
 t1\_quote: The quote as it appears in the first text  
 t2\_quote: the quote as it appears in the second text  
 '''  
   
 with open(f'data/alignments/{alignment\_file}','r',encoding='utf8') as rf:  
 intertext\_data = rf.read().split("\n")  
 # remove the header line  
 intertext\_data = intertext\_data[1:]  
 return [d.split("\t") for d in intertext\_data]  
  
def get\_document(filename):  
 '''  
 helper function to load document  
 '''  
 with open(f'data/corpus/{filename}.txt', 'r', encoding='utf8') as rf:  
 return rf.read()  
  
def get\_text\_and\_labels(input\_file\_ids, input\_file\_labels,   
 intertext\_data, n\_random\_sections,   
 random\_length\_range, balance\_samples=True):  
 '''  
 function to extract texts and labels from two files, using alignment data to block out  
 sections shared between the two works  
 '''  
   
 # limit intertext data to the current docs  
 intertext\_data = [d for d in intertext\_data if d[0] in input\_file\_ids and d[1] in input\_file\_ids]  
   
  
   
 texts = []  
 labels = []  
  
 # iterate through the files  
 for file\_id, file\_label in zip(input\_file\_ids, input\_file\_labels):  
 # fetch text  
 text = get\_document(file\_id)  
  
   
   
 # check the doc id for intertext info  
 if file\_id == intertext\_data[0][0]:  
 quote\_loc = 4  
 else:  
 quote\_loc = 5  
  
 # get the blocked quote locations working from back to front  
 blocked = sorted([[int(d[quote\_loc]), int(d[quote\_loc]) + len(d[quote\_loc+2].replace(" ", ""))]   
 for d in intertext\_data],reverse=True)  
 # remove shared text from document  
 for block in blocked:  
 text = text[:block[0]] + text[block[1]:]  
   
 # get random sections for building model  
 text\_sections = get\_random\_sections(text, n\_random\_sections, random\_length\_range)  
 section\_labels = [file\_label for \_ in text\_sections]  
 texts.extend(text\_sections)  
 labels.extend(section\_labels)  
  
 # to ensure the two works are equally represented, I make sure that no one text dominates the  
 # training set  
 if balance\_samples:  
 all\_counts = [labels.count(l) for l in input\_file\_labels]  
 if min(all\_counts) < n\_random\_sections:  
 for l in input\_file\_labels:  
 while labels.count(l) > min(all\_counts):  
 eject = labels.index(l)  
 texts.pop(eject)  
 labels.pop(eject)  
   
 return texts, labels

'''  
Run the process on the Jingshi yinyang meng and the Yujing xintan  
'''  
# Jingshi yinyang meng  
jingshi\_id = "9785"  
jingshi\_label = "Jingshi"  
  
# Yujing xintan  
yujing\_id = "18077"  
yujing\_label = "Yujing"  
  
# get intertext data  
yjjs\_intertext\_data = get\_alignment\_data("yujingjingshi.tsv")  
  
# Load and then randomly divide the two texts  
yjjs\_sections, yjjs\_labels = get\_text\_and\_labels([jingshi\_id, yujing\_id],   
 [jingshi\_label, yujing\_label],   
 yjjs\_intertext\_data,   
 n\_sections, length\_range)   
  
if verbose:  
 # print information:  
 print("sections from Jingshi")  
 print(yjjs\_sections[:2], yjjs\_labels[:2])  
 print("sections from Yujing")  
 print(yjjs\_sections[-2:], yjjs\_labels[-2:])

I then transform these substrings into vectors by calculating their vectorized representation based on *n*-gram frequency.

# uncomment the next line to install sklearn if not installed   
# %pip install scikit-learn  
from sklearn.feature\_extraction.text import TfidfVectorizer  
  
def vectorize\_texts(texts, \*\*kwargs):  
 '''  
 function that sets up and fits a vectorizer then transforms the input corpus  
  
 kwargs passes the vectorizor options from the outer function to the tfidf vectorizer  
 '''  
  
 # initialize vectorizer  
 vectorizer = TfidfVectorizer(analyzer="word", token\_pattern=".", \*\*kwargs)  
  
 # fit and transform the data  
 frequency\_vectors = vectorizer.fit\_transform(texts)  
  
 # return both vectorizer and vectors  
 return vectorizer, frequency\_vectors.toarray()

'''  
Run the vectorization process  
'''  
yjjs\_vectorizer, yjjs\_vectors = vectorize\_texts(yjjs\_sections,   
 max\_features=max\_features,   
 use\_idf=use\_idf,   
 ngram\_range=ngram\_range)  
  
# if the notebook is set to run in verbose mode then this will print out the first of the vectors.  
if verbose:  
 print(yjjs\_vectors[:1])

I use the resulting vectors to train and test a text classifier according to standard machine learning protocols, following the procedure as outlined on the scikit learn website ((Cross-Validation: Evaluating Estimator Performance, n.d.)). Once I am confident that the model performs well at determining the origin of the randomly selected substrings, and ensure that the model is not overfit, I apply the model to actual intertextual material to determine the most likely source text.

There are many types of models useful for building text-classifiers. Support vector machines (SVM) are popular for text classification tasks, so I train SVM classifiers with stochastic gradient descent in my experiments here (via the SGDClassifier class ((Sklearn.Linear\_model.SGDClassifier, n.d.))). I could spend significant time trying to identify the absolute best algorithm to use for this, but SVMs prove effective and are very efficient to train and use. Pushing accuracy to the limits, while a very valid pursuit, is outside the scope of this particular article.

from sklearn.linear\_model import SGDClassifier  
from sklearn.model\_selection import cross\_val\_score, train\_test\_split  
from sklearn.metrics import confusion\_matrix, accuracy\_score  
from IPython.display import Markdown  
import time  
  
  
def train\_and\_test\_model(frequency\_vectors, labels, test\_size=0.25, return\_scores=False, print\_results=True):  
 '''  
 function that takes vectors and labels as input data, runs cross validation, trains a final model  
 and returns the classifier  
  
 note that I am training the classifiers many times for the sake of these experiments. This would   
 not necessarily be necessary when deploying this in the course of exploratory research  
 '''  
   
 # split data into training and testing sections following sklearn conventions  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(frequency\_vectors, labels, test\_size=0.25)  
  
 # set up classifier  
 clf = SGDClassifier(loss="hinge",max\_iter=50000, random\_state=random\_seed)  
  
 # perform n fold cross validation on the training set and print results  
 scores = cross\_val\_score(clf, X\_train, y\_train, cv=5)  
  
 # retrain classifier for confusion matrix  
 clf = SGDClassifier(loss="hinge",max\_iter=50000, random\_state=random\_seed)  
 clf.fit(X\_train, y\_train)  
  
 # predict on test data  
 y\_pred = clf.predict(X\_test)  
  
 unique\_labels = list(set(labels))  
   
 if print\_results:  
 display(Markdown(f"Cross Validation scores: {scores.mean():.2f} accuracy with a standard deviation of {scores.std():.2f}"))  
  
 # print confusion matrix and accuracy scores  
 res = confusion\_matrix(y\_test, y\_pred)  
  
 md\_string = "Confusion Matrix|Actual \_"  
 md\_string += "\_|Actual \_".join(unique\_labels)  
 md\_string += "\_\n"  
 md\_string += "|".join(["---" for \_ in range(len(unique\_labels)+1)])  
 md\_string += "\nPredicted "  
 for i, r in enumerate(res):  
 r = [str(item) for item in r]  
 md\_string += unique\_labels[i] + "|"  
 md\_string += "|".join(list(r))  
 if i < len(res) - 1:  
 md\_string += "\nPredicted "  
   
 # print(res)  
 # md\_string = f"Confusion Matrix|Actual \_{unique\_labels[0]}\_|Actual \_{unique\_labels[1]}\_\n---|---|---\nPredicted {unique\_labels[0]}|{res[0][0]}|{res[0][1]}\nPredicted {unique\_labels[1]}|{res[1][0]}|{res[1][1]}"  
 display(Markdown(md\_string))  
   
  
 # retrain classifier on full training data for best performance  
 clf = SGDClassifier(loss="hinge",max\_iter=50000,random\_state=random\_seed)  
 clf.fit(frequency\_vectors, labels)  
  
   
   
 if return\_scores:  
 return clf, scores  
  
 return clf

# Train and test classifer  
yjjs\_clf = train\_and\_test\_model(yjjs\_vectors, yjjs\_labels, print\_results=verbose)

## A proof of concept

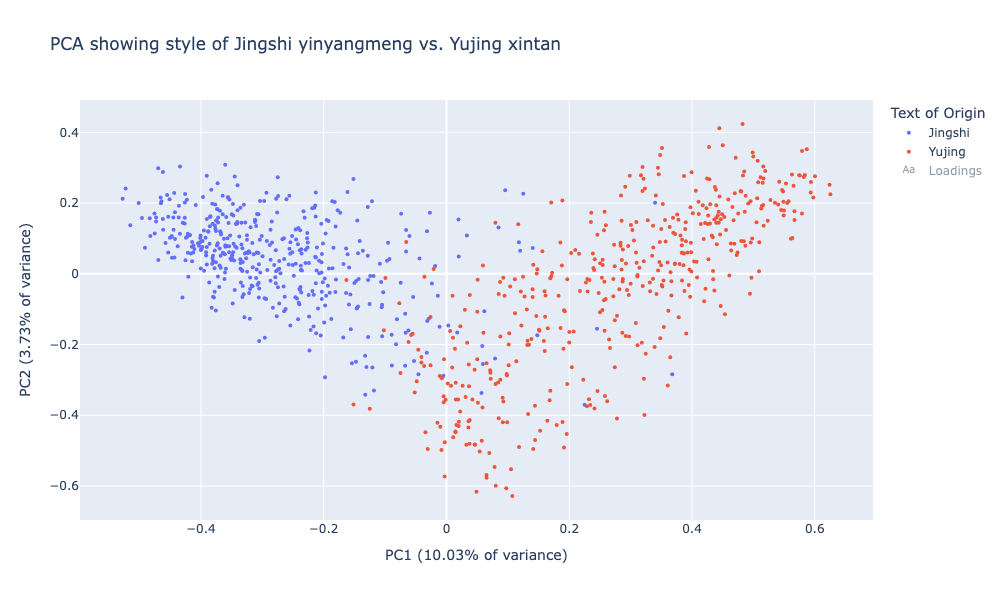
In order to assess the potential of this process, I start with a very unambiguous case of intertextuality that is clearer than most cases found within *Jinpingmei*. Two works from 1628, the *Yujing xintan* 玉鏡新譚 and the *Jingshi yinyangmeng* 警示陰陽夢 offer a good starting point. These two pieces both discuss Wei Zhongxian 魏忠賢 (1568--1627), the infamous eunuch who acquired significant political power during the Tianqi Emperor’s reign (1620--1627). The *Yujing xintan* is an unofficial history and the earliest extant work to discuss Wei Zhongxian after his death. The *Jingshi yinyangmeng* appeared a few months after the *Yujing xintan* and is essentially a novelization of the earlier history. Given the very clear textual connection between these two works, and their relatively distinct styles, the case offers a nice proof of concept. This distinct style is evident when I use principal component analysis (PCA) to visualize the variance among randomly selected substrings drawn from these two works, as shown in Figure 4.

Note: For a detailed engagement with the usefulness of principal component analysis for understanding Chinese stylistics, see ((Vierthaler, 2016)).

# uncomment the next line if pandas or plotly are not installed  
# %pip install pandas plotly  
  
from sklearn.decomposition import PCA  
import pandas as pd  
import plotly.express as px  
from plotly import graph\_objects as go  
from plotly import offline  
import plotly as py  
  
def generate\_PCA\_viz(frequency\_vectors, labels, vectorizer, title):  
 '''  
 function that takes frequency vectors and a vectorizer object and generates a principal component  
 analysis visualization.  
 '''  
   
 pca = PCA(n\_components=2)  
 my\_pca = pca.fit\_transform(frequency\_vectors)  
  
   
 pca\_df = pd.DataFrame({"pc1":my\_pca[:,0], "pc2":my\_pca[:,1],"labels":labels})  
 fig = px.scatter(pca\_df, x="pc1", y="pc2", color="labels", title=title,  
 labels={"pc1":f"PC1 ({pca.explained\_variance\_[0]\*100:.2f}% of variance)",  
 "pc2":f"PC2 ({pca.explained\_variance\_[1]\*100:.2f}% of variance)",  
 "labels":"Text of Origin"})  
 vocab = vectorizer.get\_feature\_names\_out()  
 loadings = pca.components\_  
 loadings\_df = pd.DataFrame({"vocab":list(vocab), "pc1":loadings[0], "pc2":loadings[1]})  
  
 fig.add\_trace(go.Scatter(x=loadings\_df["pc1"], y=loadings\_df["pc2"], text=loadings\_df["vocab"], mode="text", visible="legendonly", name="Loadings"))  
 fig.update\_layout(width=1000, height=600)  
 fig.update\_traces(marker={'size': 4})  
 fig.show()

# Generate Figure, to see loadings simply click "loadings" in the figure legend  
# generate\_PCA\_viz(yjjs\_vectors, yjjs\_labels, yjjs\_vectorizer,   
# "PCA showing style of Jingshi yinyangmeng vs. Yujing xintan")

from IPython.display import Image, display  
  
metadata={  
 "jdh":{  
 "module":"object",  
 "object":{  
 "type":"image",  
 "source":[  
 "PCA showing style of Jingshi yinyangmeng vs. Yujing xintan"  
 ]  
   
 }  
 }  
}  
  
display(Image("media/pcajingshivsyujing.png", width=1000), metadata=metadata)



There is some overlap in the central part of this figure when sections from each text appear in similar spaces, but it is still distinct. The component loadings also offer an opportunity to study why texts get pulled in the direction they do. The further a character is from the center of the figure (0,0) the more it “pulls” documents in that direction when it occurs frequently in the document. These first two components only show a combined thirteen percent of the total variance within the dataset. As such, PCA obscures much of the actual differences between these two works. Even so, there is a clear visual distinction between substrings from the two texts.

I use these same vectors, untransformed by PCA, to train a classifier that yields much higher discriminatory power. I begin by dividing the randomly selected substrings into two groups. First, I use 75 percent of the text vectors as a training set for the model and hold out 25 percent of them to test it. Then I use tenfold cross-validation on the training vectors to measure the general accuracy of potential classifiers trained on these materials (in this case, the ten classifiers are 98 percent accurate with a standard deviation of 2 percent). I then train a classifier on all 75 percent of the training vectors and test it against the 25 percent of held-out vectors. This results in a model that is 98 percent accurate, as shown in the confusion matrix below:

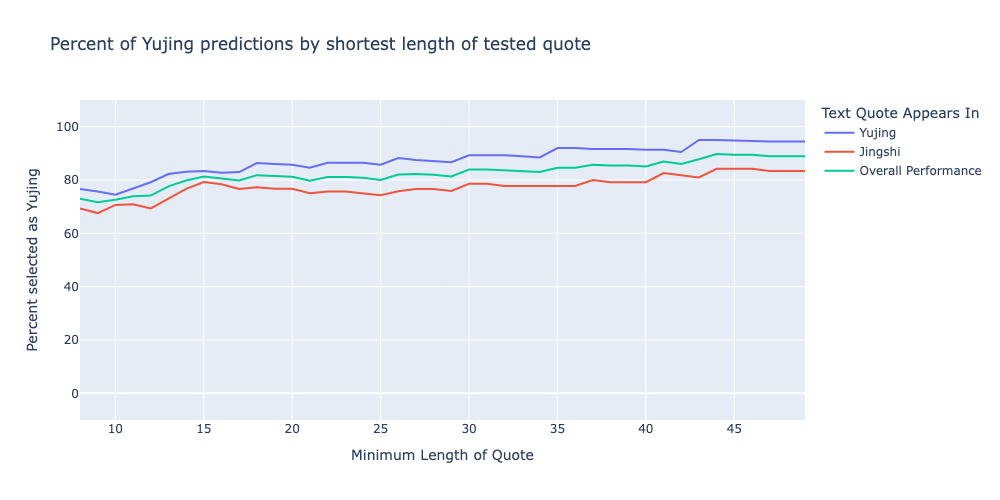
|  |  |  |
| --- | --- | --- |
| Confusion Matrix | Actual *Yujing* | Actual *Jingshi* |
| Predicted *Yujing* | 122 | 4 |
| Predicted *Jingshi* | 2 | 122 |

I then apply this model to the sections of text shared between the *Yujing xintan* and *Jingshi yinyangmeng*. I know that all the shared text originates in the *Yujing xintan*, so it is simple to assess the model's accuracy. Figure 5 shows that the model accurately traces the origins of the shared quotes to *Yujing xintan*, and the longer the quote is the better the model works.

# Constrain the shortest quotes to ever longer sections and test impact on model's accuracy.  
  
  
def get\_shared\_info(file\_ids, file\_labels, intertext\_data, limit=None):  
 '''  
 This function extracts the quotes from the intertextuality results file  
 '''  
   
 shared\_info = []  
 for file\_id, file\_label in zip(file\_ids, file\_labels):  
 for d in intertext\_data:  
 if d[0] in file\_ids and d[1] in file\_ids:  
 quote\_location = 6  
 if d[1] == file\_id:  
 quote\_location = 7  
 shared\_info.append([d[quote\_location].replace(" ", ""), file\_label])  
 if limit:   
 return [s for s in shared\_info if len(s[0]) >= limit]  
 return shared\_info  
  
def run\_multiple\_models(file\_ids,  
 file\_labels,   
 vectorizer,   
 classifier,  
 intertext\_data,   
 known\_source,  
 title,  
 start\_limit=8,   
 end\_limit=50):  
 '''  
 Function to test multiple models against varying string lengths  
 '''   
   
 #empty lists to gather info  
 limits = []  
 accuracy = []  
 samples = []  
 origins = []  
   
 for limit in range(start\_limit, end\_limit):  
 # extract just the shared info where the quote at least meets the length threshold  
 shared\_info = get\_shared\_info(file\_ids, file\_labels, intertext\_data, limit)  
  
  
 # print(shared\_info[0], shared\_info[-1])  
  
 for origin in [file\_labels[0], file\_labels[1], "Overall Performance"]:  
 # get shared text by origin  
 if origin == "Overall Performance":  
 shared\_text = [d[0] for d in shared\_info]  
 else:  
 shared\_text = [d[0] for d in shared\_info if d[1] == origin]  
  
 # get shared vectors  
 shared\_frequencies = vectorizer.transform(shared\_text).toarray()  
   
  
  
   
 # get predictions  
 shared\_pred = classifier.predict(shared\_frequencies)  
 # create results dictionary  
 results = {file\_labels[0]:0, file\_labels[1]:0}  
 for p in set(shared\_pred):  
 results[p] = list(shared\_pred).count(p)  
   
   
 if "Jinpingmei" not in results:  
 results["Jinpingmei"] = 0  
 if known\_source:  
 correct\_res = results[known\_source]  
 else:  
 # if no known source, set 1 to JPM and 0 to other.  
 correct\_res = results["Jinpingmei"]  
   
 # append results to lists  
 limits.append(limit)  
 accuracy.append((correct\_res/len(shared\_pred))\*100)  
 samples.append(len(shared\_pred))   
 origins.append(origin)   
  
 # create a dataframe for model accuracy  
 df = pd.DataFrame({"length":limits, "accuracy":accuracy, "sample\_length":samples, "origin":origins})  
  
 # visualize the model's accuracy  
 # if known\_source set accuracy label  
 if known\_source:  
 accuracy\_label = f"Percent selected as {known\_source}"  
 else:  
 accuracy\_label = f"Percent selected as {label\_1}"  
   
 fig = px.line(df, x="length", y="accuracy", color="origin",   
 labels={  
 "length":"Minimum Length of Quote",  
 "accuracy":accuracy\_label,  
 "origin":"Text Quote Appears In",  
 "sample\_length":"Total number of quotes analyzed"  
 },  
 hover\_data=["sample\_length"],   
 title=title)  
 fig.update\_layout(yaxis\_range=[-10,110], width=1000, height=500)  
 fig.show()

# specify the known source for model evaluation  
yjjs\_known\_source = "Yujing"  
   
# Generate Figure 5  
# run\_multiple\_models([yujing\_id, jingshi\_id],   
# [yujing\_label, jingshi\_label],   
# yjjs\_vectorizer, yjjs\_clf,   
# yjjs\_intertext\_data, yjjs\_known\_source,  
# "Percent of Yujing predictions by shortest length of tested quote")

from IPython.display import Image, display  
  
metadata={  
 "jdh":{  
 "module":"object",  
 "object":{  
 "type":"image",  
 "source":[  
 "Percent of Yujing predictions by shortest length of tested quote"  
 ]  
   
 }  
 }  
}  
  
display(Image("media/yujingpredbylength.png", width=1000), metadata=metadata)



The above figure shows that the accuracy of the model (shown along the y axis) improves as the minimum length of quote analyzed increases (as shown on the x-axis). This improvement in accuracy as quotes get longer is unsurprising, given that the short phrases tend not to be what I would consider meaningful sharing. Instead, these tend to be things like dates, which generate multiple instances of detected intertextuality. For example “the sixth day of the eleventh month of the seventh year of the Tianqi reign 天启七年十一月初六日 (December 13, 1627),” appears in various places but the model is not good at associating it with a specific document, nor should we expect it to be, as the phrase in question does not reflect the style of the work.

This figure demonstrates that the models are biased toward the document in which a given quote appears. This is unsurprising, given that the intertextuality algorithm allows up to 20 percent difference between strings, and quotes that an author edited to fit within their own work appear in the results. We would expect the author of the *Jingshi yinyangmeng* to edit some quotations they take from the *Yujing xintan* to better fit the style of the *Jingshi*). This editing often has a noticeable impact on model performance. This is also why I run the analysis on a per-quote basis rather than amalgamating all shared text together. I can analyze why certain specific quotes are likely to be judged one way or another to better understand the errors in the model. In the end, the model is still very accurate and convincingly establishes that the shared material originates from the *Yujing xintan*.

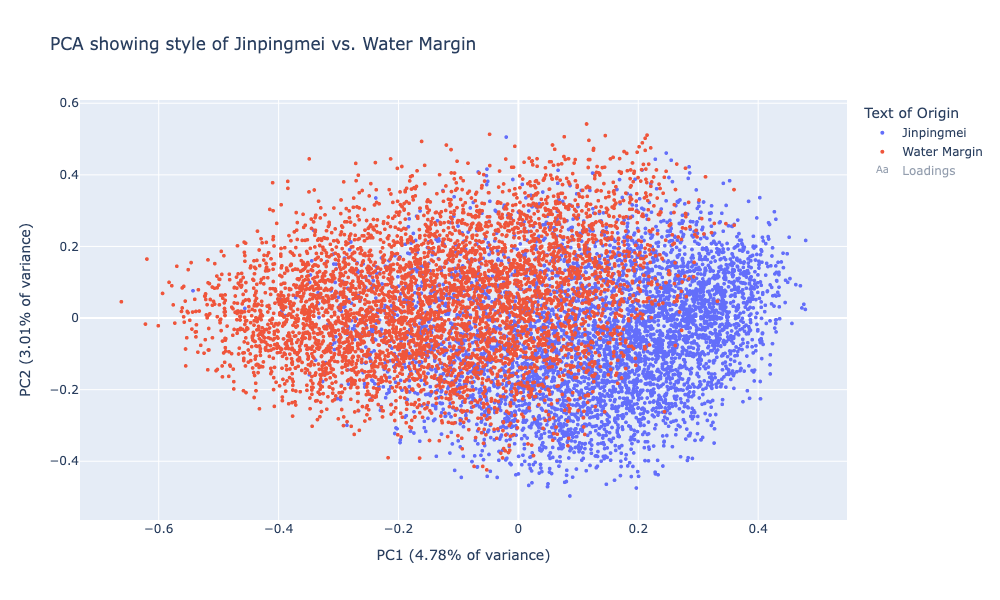
## *Jinpingmei* vs. *Water Margin*

The proof of concept above illustrates that under ideal conditions, it is possible to identify the origins of intertextuality based on apparent textual similarity. The most important intertextual cases within *Jinpingmei*, however, sometimes present us with less ideal circumstances. As I noted earlier in this article, idle readers and scholars alike have noted that *Water Margin* played an enormously influential role in the development of *Jinpingmei*. As such, the procedure I outlined above for the Wei Zhongxian texts will probably be effective, but the shared generic features of *Jinpingmei* and *Water Margin* may complicate matters. These are both novels and do not have the distinct generic features that effectively divide the *Yujing xintan* from the *Jingshi yinyangmeng*. As Figure 6 shows, when I visualize the stylistic difference among randomly selected document vectors from *Jinpingmei* and *Water Margin*, there is much more overlap between the two texts. This suggests that their distributions of frequent terms are more similar to each other than what is seen in the of the Wei texts.

# Figure 6 generation  
jpm\_id = "25272"  
jpm\_label = "Jinpingmei"  
  
shuihu\_id = "25124"  
shuihu\_label = "Water Margin"  
  
# get intertext data  
jpm\_intertext\_data = get\_alignment\_data("jpm\_alignment.tsv")  
  
# Load and then randomly divide the two texts  
jpmsh\_sections, jpmsh\_labels = get\_text\_and\_labels([jpm\_id, shuihu\_id],   
 [jpm\_label, shuihu\_label],   
 jpm\_intertext\_data, n\_sections,  
 length\_range)   
  
# Vectorize the texts  
jpmsh\_vectorizer, jpmsh\_frequency\_vectors = vectorize\_texts(jpmsh\_sections,  
 max\_features=max\_features,   
 use\_idf=use\_idf,   
 ngram\_range=ngram\_range)

# Generate figure  
# generate\_PCA\_viz(jpmsh\_frequency\_vectors, jpmsh\_labels, jpmsh\_vectorizer,   
# "PCA showing style of Jinpingmei vs. Water Margin")

from IPython.display import Image, display  
  
metadata={  
 "jdh":{  
 "module":"object",  
 "object":{  
 "type":"image",  
 "source":[  
 "PCA showing style of Jinpingmei vs. Water Margin"  
 ]  
   
 }  
 }  
}  
  
display(Image("media/pcajpmvswatermargin.png", width=1000), metadata=metadata)



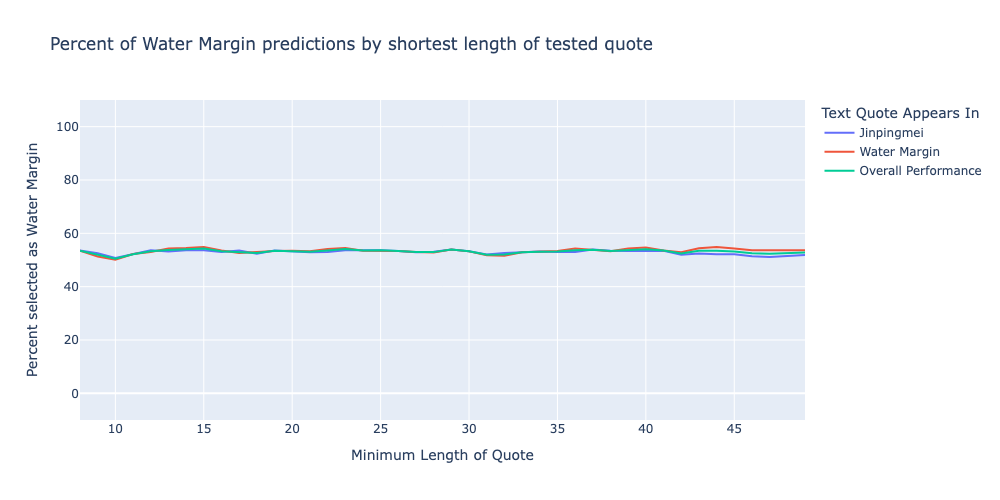
Again, significant variance hids from view in these types of visualizations. Here the first two principal components only capture the first 4.8 and 3 percent of variance respectively, much lower than the variance I capture in the *Yujing* vs. *Jingshi* comparison. But the text classifier does not operate on the principal components and can leverage all of the untransformed data. Somewhat surprisingly, the *Jinpingmei* vs. *Water Margin* model’s accuracy is admirably high. Using ten-fold cross validation, the models are 91 percent accurate. The following confusion table illustrates the results of applying the model to the validation set:

|  |  |  |
| --- | --- | --- |
| Confusion Matrix | Actual *Jinpingmei* | Actual *Water Margin* |
| Predicted *Jinpingmei* | 1127 | 122 |
| Predicted *Water Margin* | 100 | 1151 |

Yet despite the excellent performance of the model, its accuracy is much less impressive when applied to the sections of text shared between the two novels. The classifier struggles to stay much about 53 percent accuracy, and the edited sections of text found within *Jinpingmei* are associated with *Shuihu zhuan* just over 50 percent of the time. This is only marginally better than a random guess.

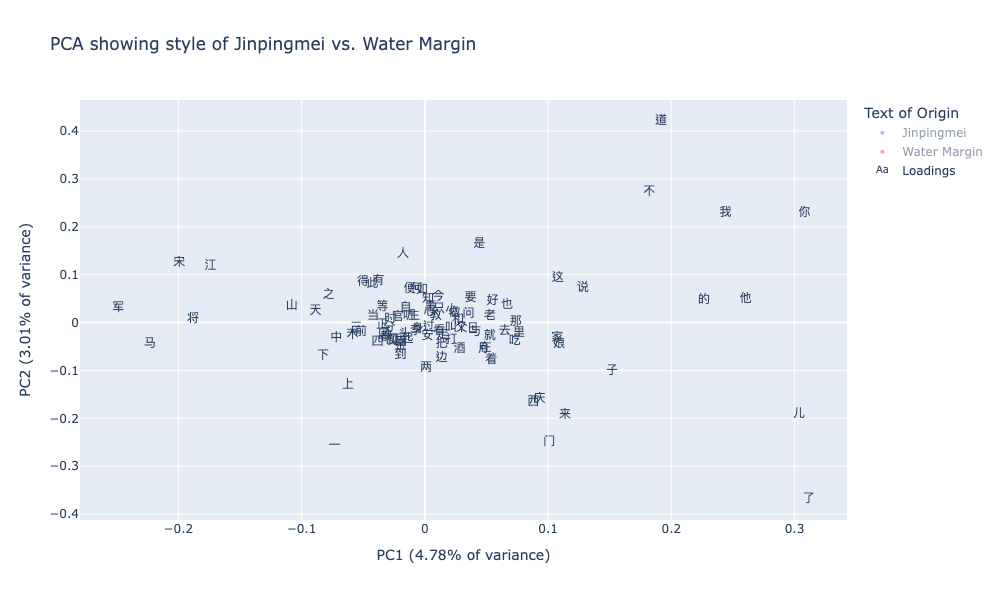
# Generate Figure 7  
# train and test the model  
jpmsh\_clf = train\_and\_test\_model(jpmsh\_frequency\_vectors, jpmsh\_labels, test\_size=0.25, print\_results=False)  
  
# Specify the known source for model evaluation  
jpmsh\_known\_source = "Water Margin"  
# run\_multiple\_models([jpm\_id, shuihu\_id],  
# [jpm\_label, shuihu\_label],   
# jpmsh\_vectorizer, jpmsh\_clf,   
# jpm\_intertext\_data,   
# jpmsh\_known\_source,  
# "Percent of Water Margin predictions by shortest length of tested quote")

from IPython.display import Image, display  
  
metadata={  
 "jdh":{  
 "module":"object",  
 "object":{  
 "type":"image",  
 "source":[  
 "Percent of Water Margin predictions by shortest length of quote"  
 ]  
   
 }  
 }  
}  
  
display(Image("media/watermarginpredbylength.png", width=1000), metadata=metadata)



The overall results of the model do point us to *Water Margin* as the likely progenitor of the shared materials, but the source of inaccuracies in the model is also very instructive. The large amount of error largely stems from the complex relationship between the two texts. First, we can expect some noise simply because the exact version of the *Water Margin* the author likely used to compose *Jinpingmei* is not present in the corpus (and does not seem to exist anymore). Additionally, *Jinpingmei* is a novel centered on several important people who also appear in *Water Margin*. Most prominent is Ximen Qing 西門慶, the anti-hero of *Jinpingmei*. He dominates the narrative for the first 79 chapters of *Jinpingmei*, at which point he dies. Pan Jinlian 潘金蓮, the woman whose wooing occupies the first section of *Jinpingmei* and whose presence dominates the narrative of the novel as a whole, also appears in *Water Margin*. In *Water Margin*, Wu Song kills Ximen Qing and Pan Jinlian at the offset of their relationship. As such, Ximen and Pan only appear in a few chapters of *Water Margin*. But *Jinpingmei’s* author copies these chapters nearly verbatim but alters the story such that Ximen Qing and Pan Jinlian initially escape Wu’s wrath. They each go on to appear in dozens of chapters. Given this, a randomly selected piece of text from *Water Margin* is not likely to mention either figure. On the other hand, random text from *Jinpingmei* is very likely to mention one or both of them. Furthermore, nearly all of their mentions in *Water Margin* are concentrated in intertextual moments, so the model tends to associate them with *Jinpingmei*. This phenomenon appears in the loadings of the *Jinpingmei* vs. *Water Margin* PCA, where the Chinese characters in Ximen Qing, Pan Jinlian, and select other peoples’ names play an important role in distinguishing the two novels.

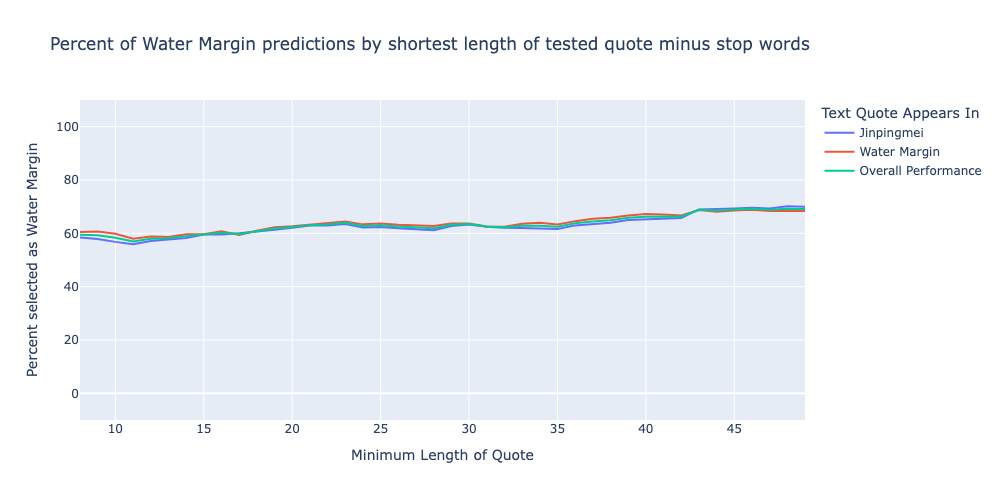
from IPython.display import Image, display  
  
metadata={  
 "jdh":{  
 "module":"object",  
 "object":{  
 "type":"image",  
 "source":[  
 "Loadings from PCA showing style of Jinpingmei vs. Water Margin"  
 ]  
   
 }  
 }  
}  
  
display(Image("./media/pcajpmvswatermarginloadings.png", width=1000), metadata=metadata)



As such, the model is biased toward evaluating any quote that mentions either of these characters as original to *Jinpingmei*. If I remove the *n*-grams within Ximen Qing and Pan Jinlian’s names from the corpus prior to training, then the accuracy of the model moves well into the 70s, as shown in the figure below. Rather than being detrimental to the efficacy of this approach, it reveals the necessity of understanding exactly what causes these models to decide things in the way that they do.

# Code needed to generate figure 8  
  
# Revectorize the texts  
# filter out characters from jpm  
jpm\_stopwords = ["西", "门", "庆", "潘", "金", "莲", "吴", "月", "娘", "李"]  
jpmsh\_vectorizer, jpmsh\_frequency\_vectors = vectorize\_texts(jpmsh\_sections,  
 max\_features=max\_features,   
 use\_idf=use\_idf,   
 ngram\_range=ngram\_range,  
 stop\_words=jpm\_stopwords)  
  
# Specify the known source for model evaluation  
jpmsh\_known\_source = "Water Margin"  
# run\_multiple\_models([jpm\_id, shuihu\_id],  
# [jpm\_label, shuihu\_label],   
# jpmsh\_vectorizer, jpmsh\_clf,   
# jpm\_intertext\_data,   
# jpmsh\_known\_source,  
# "Percent of Water Margin predictions by shortest length of tested quote minus stop words")

from IPython.display import Image, display  
  
metadata={  
 "jdh":{  
 "module":"object",  
 "object":{  
 "type":"image",  
 "source":[  
 "Percent of Water Margin predictions by shortest length of quote minus stop words"  
 ]  
   
 }  
 }  
}  
  
display(Image("./media/watermarginpredbylengthnostopwords.png", width=1000), metadata=metadata)

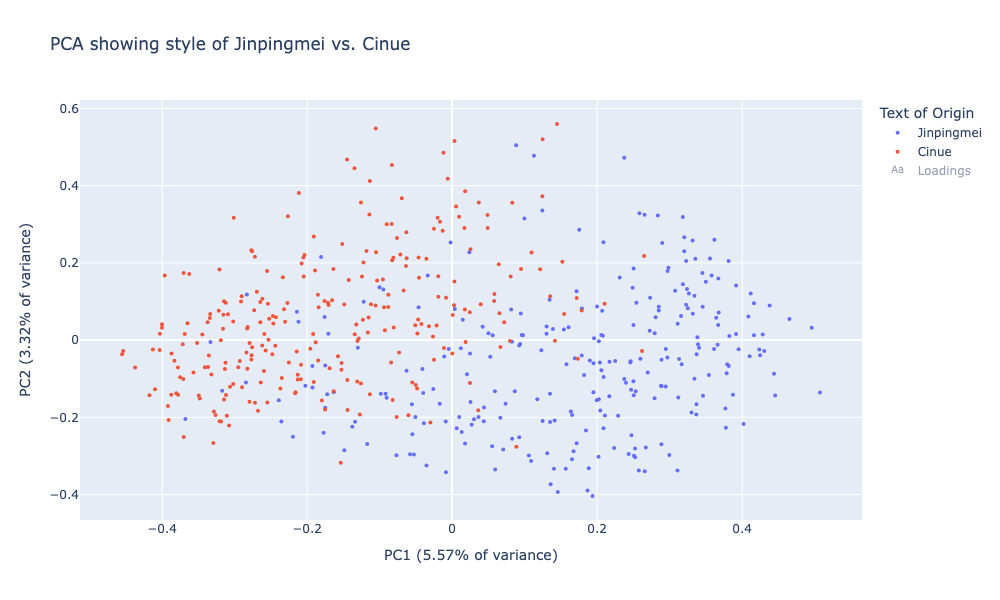


## Jinpingmei vs. Li Kaixian’s Cinüe

Another intriguing example of intertextuality exists between *Jinpingmei* and Li Kaixian’s work *Cinüe* 詞謔, a rather obscure piece of dramatic criticism. These two works share a smattering of text throughout *Jinpingmei*, but one scene of particular importance shared between the two occurs in chapter 71 of *Jinpingmei*, in which Ximen Qing and He Xin the Eunuch Director listen to a rather extended song suite (for an English translation see ((Roy, 2011), 309)). Establishing the directionality of quotation is particularly interesting if it turns out the *Cinüe* is quoting from *Jinpingmei*; this might then serve as evidence that Li Kaixian is a prime candidate for the likely identity of the author of *Jingpingmei*. Some scholars already suspect Li may be the author, but he passed away in 1568, well before there is evidence of *Jinpingmei* circulating. Beyond its implications for authorship, this would also potentially push the earliest date of composition of the novel back in time substantially.

# Figure 8 generation  
cinue\_id = "21285"  
cinue\_label = "Cinue"  
  
# Load and then randomly divide the two texts  
jpmcn\_sections, jpmcn\_labels = get\_text\_and\_labels([jpm\_id, cinue\_id],   
 [jpm\_label, cinue\_label],   
 jpm\_intertext\_data, n\_sections,length\_range)   
  
# Vectorize the texts  
jpmcn\_vectorizer, jpmcn\_frequency\_vectors = vectorize\_texts(jpmcn\_sections,  
 max\_features=max\_features,   
 use\_idf=use\_idf,   
 ngram\_range=ngram\_range,  
 stop\_words=jpm\_stopwords)  
  
# Generate Figure 8  
# generate\_PCA\_viz(jpmcn\_frequency\_vectors, jpmcn\_labels, jpmcn\_vectorizer,   
# "PCA showing style of Jinpingmei vs. Cinue")

from IPython.display import Image, display  
  
metadata={  
 "jdh":{  
 "module":"object",  
 "object":{  
 "type":"image",  
 "source":[  
 "PCA showing style of Jinpingmei vs Cinue"  
 ]  
   
 }  
 }  
}  
  
display(Image("media/pcajpmcn.png", width=1000), metadata=metadata)



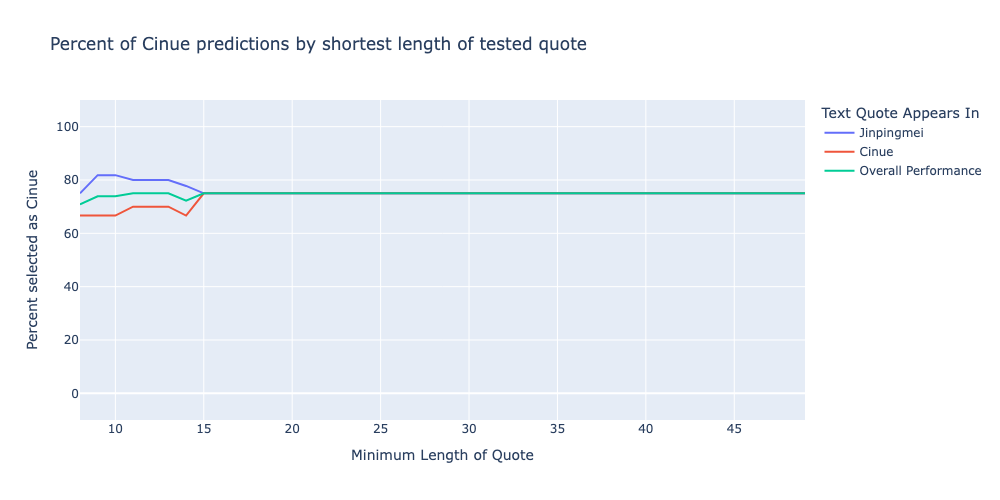
This figure shows that the style of *Jinpingmei* and *Cinüe* is distinct. There is some amount of stylistic overlap, but not nearly so much as seen between *Jinpingmei* and *Water Margin*. Here, the algorithm produces a model that is 94 percent accurate with a standard deviation of 3 percent. The fully trained model is 95 percent accurate and results in the following confusion matrix.

|  |  |  |
| --- | --- | --- |
| Confusion Matrix | Actual *Jinpingmei* | Actual *Cinüe* |
| Predicted *Jinpingmei* | 133 | 9 |
| Predicted *Cinüe* | 2 | 106 |

Following the same process as above I can also train and test a number of models to measure the impact the length of a quote has on model accuracy.

jpmcn\_clf = train\_and\_test\_model(jpmcn\_frequency\_vectors, jpmcn\_labels, test\_size=0.25, print\_results=False)  
  
jpmcn\_known\_source = "Cinue"  
# run\_multiple\_models([jpm\_id, cinue\_id],   
# [jpm\_label, cinue\_label],   
# jpmcn\_vectorizer, jpmcn\_clf,  
# jpm\_intertext\_data, jpmcn\_known\_source,  
# "Percent of Cinue predictions by shortest length of tested quote")

from IPython.display import Image, display  
  
metadata={  
 "jdh":{  
 "module":"object",  
 "object":{  
 "type":"image",  
 "source":[  
 "Percent of Cinue predictions by shortest length of tested quote"  
 ]  
   
 }  
 }  
}  
  
display(Image("media/cinuepredbylength.png", width=1000), metadata=metadata)



The model consistently predicts that the origin of the shared quotes are from the *Cinüe* rather than from *Jinpingmei*. While this is disappointing in terms of the potential authorship of *Jinpingmei*, as we cannot use this as evidence that Li had a copy of the novel, it does comport well with the expectations we would have upon a careful expectation of the *Cinüe* as a piece of dramatic criticism and upon considering the deeper layers of quotation that appear in these sections of *Jinpingmei*. There is also the slight complication of genre here: it may be that the algorithm focuses heavily on genre (the *Cinüe* being full of poetry and drama and *Jinpingmei* mostly consisting of prose). Given the layered nature of my approach, this doesn’t present too much of an issue in this particular case because I can trace this material further into the past, but it could potentially present roadblocks elsewhere.

## *Jinpingmei* vs. the World

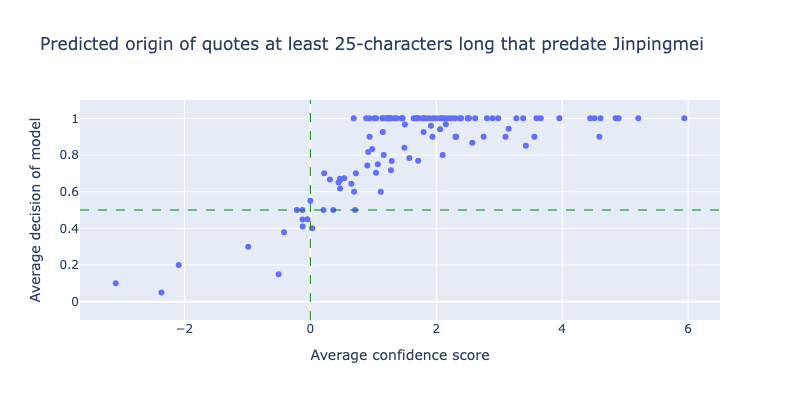
The final step I will take in evaluating this approach is to build models to compare *Jinpingmei* in a pair-wise fashion against every earlier text in the corpus. This means that I train an independent model for each comparison (so *Jinpingmei* vs. *Water Margin*, *Jinpingmei* vs. *Cinüe*, *Jinpingmei* vs. *Qingpingshan huaben* and so on). The main utility of this approach is ease of interpretability and accuracy as multi-class models become much more difficult to work with as the number of classes increase. I should note that training so many models is computationally intense, but the method remains relatively scalable as each pair-wise classifier takes somewhere between .005 seconds and 2 seconds to train in this notebook run on a 2024 MacBook Pro (I could significantly optimize if I need to perform larger-scale analysis).

In doing so, I can assess general performance and highlight drawbacks to this approach. Figure 10 shows how each of these models performs on cases of material known to predate *Jinpingmei*.

import numpy as np  
  
with open('data/jpm\_intermeta\_s.tsv', 'r', encoding='utf8') as rf:  
 jpm\_meta = rf.read().split("\n")  
jpm\_meta = [d.split("\t") for d in jpm\_meta]  
  
def one\_vs\_many\_models(file\_1, label\_1, metadata,  
 intertext\_data, n\_sections,   
 looking\_at, length\_limit, plot\_title,   
 show\_summary=True, print\_quotes=None, num\_iterations=10):  
 '''  
 This function trains multiple models comparing one text against the rest of the texts   
 that share material with it.  
  
 Reruning this multiple times may offer better insight into the likely origin of quotes  
 as the results tend to smooth out significantly.  
 '''  
   
 # get the files that are present  
 files\_in\_intertext\_data = set([d[1] for d in intertext\_data])  
   
 # limit the comparative files to just those pertaining to the looking\_at  
 # variable (before, for example, limits the texts to works that predate  
 # Jinpingmei.  
 comparative\_files = [[d[0], d[1],d[-1]] for d in metadata if d[-1] in looking\_at and d[0] in files\_in\_intertext\_data]  
   
 all\_results = {}  
   
 pred\_earlier = set()  
 pred\_jpm = set()  
 miss\_classed = []  
  
 quote\_prediction = {}  
 quote\_text = {}  
 for \_ in range(num\_iterations):  
 for i,comparative\_file in enumerate(comparative\_files):  
   
 file\_2 = comparative\_file[0]  
 label\_2 = comparative\_file[1]  
 looking\_at\_state = comparative\_file[2]  
   
   
 # get the shared quotes and the text they originate in  
 shared\_info = get\_shared\_info([file\_1, file\_2],   
 [label\_1,label\_2],   
 intertext\_data,   
 limit=length\_limit)  
   
 # if there are no shared quotes (which can occur if nothing meets the length criteria)  
 # move to the next iteration  
 if len(shared\_info) == 0:  
 continue  
  
  
   
 # get text and labels to train on  
 sections, labels = get\_text\_and\_labels([file\_1, file\_2],   
 [label\_1, label\_2],   
 intertext\_data, n\_sections,  
 length\_range)   
   
   
 # limit the training to works that have at least 10 random sections each  
 if len(sections) < 20:  
 continue  
  
  
 if file\_2 not in all\_results:  
 all\_results[file\_2] = {"title":label\_2,  
 "looking\_at\_state":looking\_at\_state,  
 "acc":[],  
 "decision":[],  
 "confidence":[]}  
  
 local\_results = all\_results[file\_2]   
  
   
 # Vectorize the sections  
 vectorizer, frequency\_vectors = vectorize\_texts(sections,  
 max\_features=max\_features,   
 use\_idf=use\_idf,   
 ngram\_range=ngram\_range,  
 stop\_words=jpm\_stopwords)  
   
   
 # train and test classifier  
 clf,scores = train\_and\_test\_model(frequency\_vectors, labels, return\_scores=True, print\_results=False)  
   
 # filter down to just shared text and then create vectors  
 shared\_text = [d[0] for d in shared\_info]  
 shared\_frequencies = vectorizer.transform(shared\_text)  
   
 # make predictions annd get confidence measures of the decisions  
 shared\_pred = clf.predict(shared\_frequencies)  
 confidence\_measures = clf.decision\_function(shared\_frequencies)  
   
 local\_results["acc"].extend(scores)  
 local\_results["decision"].extend(shared\_pred)  
 local\_results["confidence"].extend(confidence\_measures)  
  
   
 # print the results from a text if interested  
 if print\_quotes and label\_2 == print\_quotes:  
 for p,s,c in zip(shared\_pred, shared\_text,confidence\_measures):  
 print(p,s,c)  
  
  
 # compile results for display:  
 compiled\_results = {"id":[], "title":[], "looking\_at\_state":[], "mean\_acc":[],  
 "avg\_decision":[], "avg\_confidence":[],  
 "num\_quotes":[]}  
  
 for doc\_id,info\_dict in all\_results.items():  
   
 # create results dictionary  
 results = {label\_1:0, info\_dict["title"]:0}  
  
 # populate results dictionary  
 for p in set(info\_dict["decision"]):  
 results[p] = info\_dict["decision"].count(p)  
  
 decision\_proprotion = results[info\_dict["title"]]/len(info\_dict["decision"])  
   
 compiled\_results["id"].append(doc\_id)  
 compiled\_results["title"].append(info\_dict["title"])  
 compiled\_results["looking\_at\_state"].append(info\_dict["looking\_at\_state"])  
 compiled\_results["mean\_acc"].append(np.array(info\_dict["acc"]).mean())  
 compiled\_results["num\_quotes"].append(len(info\_dict["confidence"])//num\_iterations)  
 compiled\_results["avg\_decision"].append(decision\_proprotion)  
 compiled\_results["avg\_confidence"].append(np.array(info\_dict["confidence"]).mean())  
   
 df = pd.DataFrame(compiled\_results)  
  
 print("total earlier text", len(df[df["avg\_decision"] > .5]))  
 print("total jpm pred", len(df[df["avg\_decision"] < .5]))  
 print("total unsure", len(df[df["avg\_decision"] == .5]))  
 fig = px.scatter(df, x="avg\_confidence", y="avg\_decision",   
 labels = {"mean\_acc":"Accuracy of classifier", "num\_quotes":"Number of quotes analyzed",  
 "title":"Title", "avg\_decision":f"Average decision of model",  
 "avg\_confidence":"Average confidence score"},  
 hover\_data=["id", "mean\_acc", "title", "num\_quotes"],  
 title=plot\_title)  
 fig.update\_layout(width=800,height=400,yaxis\_range=[-.1,1.1])  
 fig.add\_vline(x=0, line\_width=1, line\_dash="dash", line\_color="green")  
 fig.add\_hline(y=0.5, line\_width=1, line\_dash="dash", line\_color="green")  
 fig.show()  
   
 if show\_summary:  
 fig = px.histogram(df, x="avg\_confidence", labels={"count":"Total texts", "avg\_confidence":"Average Confidence Score"})  
 fig.update\_layout(width=800,height=400, yaxis\_title="Total texts")  
 fig.show()

# Figure 10 generation  
# Include only texts that are known to come before Jinpingmei  
looking\_at = ["before"]  
length\_limit = 25  
  
  
# one\_vs\_many\_models(jpm\_id, jpm\_label, jpm\_meta, jpm\_intertext\_data,  
# n\_sections, looking\_at, length\_limit,   
# "Predicted origin of quotes at least 25-characters long that predate Jinpingmei",   
# show\_summary=False)

from IPython.display import Image, display  
  
metadata={  
 "jdh":{  
 "module":"object",  
 "object":{  
 "type":"image",  
 "source":[  
 "Predicted origin of quotes at least 25-characters long that predate Jinpingmei"  
 ]  
   
 }  
 }  
}  
  
display(Image("media/predoriginpredate.png", width=1000), metadata=metadata)



In the above figure, each dot represents a comparison between all quotes shared between *Jinpingmei* and an earlier text. The y-axis shows the average decision. 0 indicates that the model predicts that all shared quotes come from *Jinpingmei* and 1 indicates that all appear to come from the opposing text, and .5 indicates that half the model’s predictions go one way and half go the other. Importantly, this is not to say that in cases where a dot falls at .5 that half the quotes *actually* come from one text and half from the other. Rather, the model is simply wrong half the time. An alternative explanation may be that in half the cases an author edited the shared quotes enough for the model to judge them as more similar to the text doing the quoting. The x-axis shows the average confidence the model has in the decisions it makes. Technically, this is the average distance the shared quotes are from the hyperplane that separates *Jinpingmei* from the other text. Negative numbers indicate the model evaluates a quote as from *Jinpingmei* and positive numbers indicate that it evaluates the quote to be from the other text. In essence, the models judge works in the lower left likely to be quoting from *Jinpingmei* and those in the upper right to be quoted by it. The total number of quotes involved in the decision also has an important impact, as the more quotes the models can operate on, the more we can rely on the results. It is far more informative if the model judges 70 percent of 400 shared quotes to be from one text than if it guesses that 100 percent of 2 quotes are from another.

When I amalgamate these results and establish that results above .5 are correct, the models work well and accurately predicts that quotes from 101 of 114 texts are more likely to be the source of the quotes than *Jinpingmei* where the evaluated quotes are at least 25 characters long. The model incorrectly predicts that ten texts quote *Jinpingmei*, and the results are evenly split for another two texts. This is decent performance but slightly lower than I expected given the performance of the models when I was conducting cross-validation.

Importantly, cases of mistaken origin are not random. Many stem from quotes shared widely across the corpus. Many mistaken quotations occur when a quote is not original to either *Jinpingmei* or the other text. Fortunately, these will often have a clearer origin elsewhere in the corpus. Genre also causes some issues, as quotations from poetry are among the more difficult to pin down. Among the missed cases include a poem found in a commentary on a Daoist text called the *Taishang laojun shuochang qingjing jingzhu* 太上老君說常清靜經注:

The beauty of sixteen has a body as smooth as cream, Her loins are a sword with which to slay the unwary. Though no one may see your head fall from your neck, Before you know it, the Marrow of your bones is sapped ((Roy, 2011), 640). 二八佳人體似酥, 腰間仗劍斬愚夫; 雖然不見人頭落, 暗裏教君骨髓枯。

The only difference between the two versions is the *Taishang laojun* version uses *fenming* ("evident" 分明) instead of *suiran* ("though" 雖然). In this case, the tone and content of the poem line up clearly with *Jinpingmei* and less so with the broader context of *Taishang laojun*.

The models also mistakenly attributes the single quotation from the *Western Chamber* (*Xixiang ji* 西廂記) in the results to *Jinpingmei*. This quote appears in the midst of a song suite that is clearly a pastiche of the types of songs present in *Western Chamber*. This possibly because there are several Chinese characters that appear in characters' names:

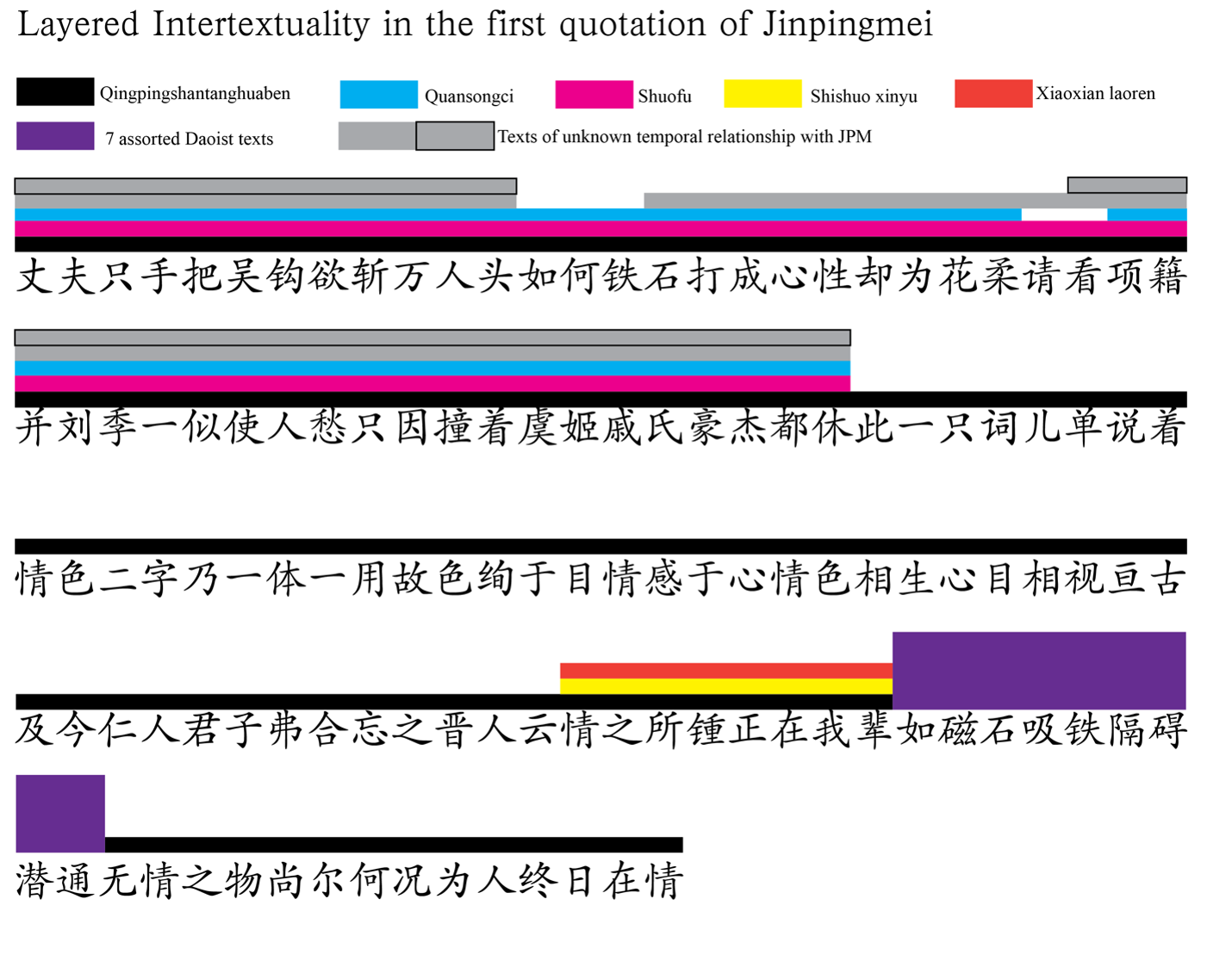
...gold lamé curtains depicting mandarin ducks in the moonlight; And folding jade screens adorned with kingfishers enjoying the breezes of spring. The music of wedding bells, Will be accompanied with phoenix flutes and ivory clappers, Patterned cithara and phoenix pipes ((Roy, 2011), 432). 鴛鴦夜月銷金帳，孔雀春風軟玉屏。合歡令,更有那鳳簫象板錦瑟鷥笙。

Despite a few clear misses, there are very few cases where this approach completely fails to identify a plausible earlier text for shared quotes that appear in earlier works, but individual cases still present some issues. The accuracy is interestingly variable when looking at different versions of the *Water Margin*. Most editions tend to sit relatively close to the decision border, with the 100 chapter edition being most likely to be mistaken for quoting *Jinpingmei*. This may point to valuable directions for future research: some editions may be closer to the one the author originally used, while others may have been produced after *Jinpingmei* and are in fact being influenced by it (although at this point this remains speculative).

The complex results illustrated in this figure highlight that one cannot treat intertextuality in a naïve fashion and simply assume that the models will be infallible. The model can only tell us, based on the input vectors, which document a string appears most similar to, even when the actual origin of the quote might be represented elsewhere in the corpus. It makes sense to find ways of increasing the fidelity of the results. One option is to train a multi-class model that aims to ascribe each quote to a particular text within the corpus. While an important step to take eventually, this is not necessarily the best first approach. As the number of classes in a model increases, the more complex, and often less reliable, the model becomes.

Instead, a useful first step in trying to trace the ultimate source of quotes within the novel might be to narrow the search space as much as possible before even building any models. One heuristic for dealing with heavily layered text is to follow Hanan’s example by looking for instances of the overlap and identifying the work where the longest matching quotes appear. If four different texts all share the same substrings, but one of them encompasses all the rest, this is likely to be the actual origin of the quote. Thus, it may not even be necessary to compare *Jinpingmei* against a text if it is clearly not the origin of the material. The quote from the *Qingpingshantang huaben* that appears at the beginning of this article (and the novel itself) is a good example: the intertextuality algorithm identifies three texts in which the opening poem appears: the *Qingpingshantang huaben*, the *Quansongci* 全宋詞, and the *Shuofu* 說郛. Yet the *Qingpingshantang huaben* quote continues beyond the poem and then later intersects with quotes from even more works, as shown in Figure 11. Ideally, I would simply compare *Jinpingmei* against *Qingpingshantang huaben* and dispense with the comparisons with the other works.

from IPython.display import Image, display  
  
metadata={  
 "jdh":{  
 "module":"object",  
 "object":{  
 "type":"image",  
 "source":[  
 "figure 11: Layered Intertextuality in the first quotation of \_Jinpingmei\_"  
 ]  
   
 }  
 }  
}  
  
display(Image("media/layeredintertextuality.png", width=1000), metadata=metadata)

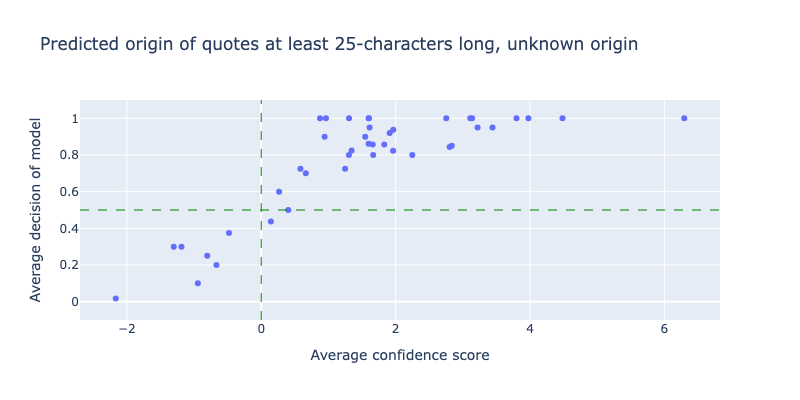


This process of assessing the nested nature of quotes can significantly narrow the unknown/contemporary quotation space that I need to test (this same sort of nestedness is [well attested to in early Chinese materials on ctext.org](https://ctext.org/tools/parallel-passages)). Keep in mind that this approach is not perfect, particularly given that the author of *Jinpingmei* may have used multiple versions of the same text to create his narrative. There are still numerous cases where quotes appear in varying lengths across many of the same texts, which can make disambiguating them very difficult. But identifying a single work to compare against *Jinpingmei* significantly eases the complexity of building a taxonomy of source materials. This is to say that instead of testing the texts represented by every nested quote in the corpus, I can ignore material embedding within longer quotes found in earlier works. In doing so, I can produce the final figure for unknown quotations within *Jinpingmei*, which fills out an initial, but sketchy, source map for the novel among contemporary works.

# simple function to filter out nested quotes (but only if they are tested within quotes known  
# to before jinpingmei)  
def filter\_nested\_quotes(intertext\_data, metadata):  
 range\_data = [(int(d[4]), int(d[4])+int(d[2]), i, metadata[d[1]][-1]) for i, d in enumerate(intertext\_data)]  
 range\_data.sort()  
  
 before\_ranges = [r for r in range\_data if r[3] == "before"]  
 other\_ranges = [r for r in range\_data if r[3] not in ["before", "after"]]  
  
 keep\_ranges = set()  
   
 # keep all before ranges  
 for before\_range in before\_ranges:  
 keep\_ranges.add(before\_range[2])  
   
 # iterate through unknown ranges (this is very inefficient but works in a pinch  
 for other\_range in other\_ranges:  
 # iterate through before ranges  
 keep = True  
 for before\_range in before\_ranges:  
 # if the other range is nested in the before range, save  
   
 if other\_range[0] >= before\_range[0] and other\_range[1] <= before\_range[1]:  
 keep=False  
 continue  
 if keep:  
 keep\_ranges.add(other\_range[2])  
   
 # return just the kept ranges  
 return [d for i,d in enumerate(intertext\_data) if i in keep\_ranges]  
  
# remove nested quotes  
jpm\_meta\_dict = {d[0]:d for d in jpm\_meta}  
no\_nested\_data = filter\_nested\_quotes(jpm\_intertext\_data, jpm\_meta\_dict)

# Figure 12 generation  
looking\_at = ["unknown","blank"]  
  
# run on unknown material  
# one\_vs\_many\_models(jpm\_id, jpm\_label, jpm\_meta,   
# no\_nested\_data,  
# n\_sections,   
# looking\_at, length\_limit,  
# "Predicted origin of quotes at least 25-characters long, unknown origin",   
# show\_summary=False)

from IPython.display import Image, display  
  
metadata={  
 "jdh":{  
 "module":"object",  
 "object":{  
 "type":"image",  
 "source":[  
 "figure 3: Heatmaps of Shared Text in Jinpingmei by Chapter in Unknown Texts"  
 ]  
   
 }  
 }  
}  
  
display(Image("media/predoriginunknown.png", width=1000), metadata=metadata)



This source map is instructive but requires some careful parsing to further our understanding of *Jinpingmei*’s sources. It shows that most contemporary materials are likely sources for the novel, and a careful study of these works will help map out a concrete network of sources. In most of the cases where the model attributes a quote to *Jinpingmei*, it is a poem with *highly* sexual content. Spending time with these results also helps me identify missing or inaccurate metadata. For example, the model tests the relationship between the *Baduanjin* 八段錦 collection of short stories and *Jinpingmei*, revealing the former’s clear dependence on the latter.

After having failed to locate their relatives, and being unable to locate a dwelling on such short notice, they asked our neighbor Old Man Fan if they could stay here for two or three days before moving on. I was planning to report this to you sir, but you have asked me about it before I was able to do so ((Roy, 2013), 357). 一時間無尋房住, 央此間鄰居範老來說, 暫住兩三日便去。正欲報知官人, 不想官人來。

This sentence is repeated verbatim with only two minor character variations in *Baduanjin* and flanked on both sides by many shorter matches that extend across multiple paragraphs. The results are fractured because the people featured in the stories are different and causing the intertextuality algorithm to return multiple shorter substrings. The *Baduanjin* story is itself likely based on a story from Feng Menglong’s 馮夢龍 *Yushi mingyan* (Stories to Enlighten the World 喻世明言), which also clearly copies from *Jinpingmei*.

In essence, I can rapidly identify the echos of *Jinpingmei* in later works and move the *Baduanjin* from the indeterminate pile into the “clearly after *Jinpingmei*” pile, despite the lack of clarity surrounding its publication history. The *Database of Premodern Chinese Popular Literature* (*Zhongguo suwen ku* 中國俗文庫) lists the *Baduanjin* as a Ming text ((Zhongguo Su Wen Ku, n.d.)), while the *Zhongguo tongsuxiaoshuo zongmu* implies it is a Qing work ((Zhongguo Tongsu Xiaoshuo Zongmu, 1990), 638). *Baduanjin* is a Qing production while others view it as a possible late Ming work.

## Conclusion

One can, and should, approach these results with some amount of suspicion. I expect the model to struggle on strings with certain characteristics. Short quotes, quotes that contain characters appearing in people’s names, and a multitude of other factors can all affect the output. The model is less reliable when the comparison text is a genre that includes very diverse types of writing. It clearly struggles to land on a consistent representation of the style of the *Yongle dadian canjuan* (*Yongle Encyclopedia Extant Sections* 永樂大典殘卷) for example. Thus rather than being an ending point, this approach opens a departure point for broader projects. Which of these quotations actually does originate in *Jinpingmei*? Can we weed out incorrect answers with careful scrutiny of the shared quotation material? After all, we can use this approach to make statements about systems of texts, but our confidence plummets when making claims about specific relationships. Would more sophisticated models clear up the inconsistencies we see here? Finally, as these results are all carefully assessed, can these predictions help us narrow down the likely date of completion of *Jinpingmei*? What might that tell us about the author of the novel? What I find particularly exciting about these results is that even in cases where the model is clearly making incorrect decisions, it is doing so in a way that tells us more about intertextual material.

While still highly experimental, this approach to deriving source materials in Jinpingmei may become widely useful in Chinese literary and historical scholarship. Like many similar methodologies, the one outlined in this article is excellent at scale: filling out the rough shapes of the corpus and source material and laying out tantalizing strings to pull at, but it still struggles with the finer philological details when the question at hand demands us to ask “is this specific result accurate?” Fortunately, this is the perfect place for scholars to apply their own expertise. And increasing the general reliability of the results is a matter of refinement (of the corpus, of its associated metadata, and of the vector representations). Importantly, it is valuable for pointing us to sections of shared text that deserve our close evaluation, a task it has already succeeded at.

There is significant utility in being able to computationally assess the nature of intertextuality, even when the complexity of the text at hand is not on par with a work like *Jinpingmei*. The likely direction of shared intertextuality can reveal significant new insights into the development of various literary, historical, and cultural phenomena. It opens more space for using network analysis and may reveal much about the intellectual community behind the productions of unusual works like Jinpingmei. This approach is particularly valuable for less-studied and less-complex works than *Jinpingmei*. As the number of novels, histories, and other materials increase through the Qing dynasty, so too do the number of works relying on multifluous prior materials. At the same time as this scope opens, our ability to pay attention to a significant percent of extant materials falls off. With tools such as these working as a guiding hand, we can start to shed light on otherwise ignored materials. And, as the extent and quality of available corpora increase and machine learning models become ever more easily accessible, approaches like this will become an important part of many scholars’ toolboxes.

## Coda

### Future aproaches

As a closing note, the code in this article is configured in such a way that building multi-class classifiers is as simple as inputting more than two texts into the get\_text\_and\_labels function. This, along with turning to LLMs for more sophisticated document representations, would be an ideal next step in testing the future of this approach. I encourage readers to experiment with this code, and the various parameters I have set above, as there are bound to be significant refinements possible with simple tuning.

### A note on the intertextuality data

Because of the complexity of the code and slow speed of processing, I have included the output of the intertextuality algorithm in the data folder rather than including the code here. I slightly modified the code linked in Vierthaler and Gelein and further developed it to run on config files instead of directly inputting the variables into the code. To recreate the full intertextual results, one simply needs to [run the code found here](https://github.com/vierth/intertextualityjpm).

# The code I have produced makes designing a multiclass classifier easy  
text\_ids = ["25272", "42420", "25124"]  
text\_labels = ["Jinpingmei", "Water Margin (70 chapter)", "Water Margin"]  
  
multiple\_sections, multiple\_labels = get\_text\_and\_labels(text\_ids,   
 text\_labels,   
 jpm\_intertext\_data,   
 n\_sections, length\_range,  
 balance\_samples=True)   
  
# Vectorize the texts  
multiple\_vectorizer, multiple\_frequency\_vectors = vectorize\_texts(multiple\_sections,  
 max\_features=max\_features,   
 use\_idf=use\_idf,   
 ngram\_range=ngram\_range,  
 stop\_words=jpm\_stopwords)  
  
# Visualize relationshipos  
# generate\_PCA\_viz(multiple\_frequency\_vectors, multiple\_labels, multiple\_vectorizer,   
# "PCA comparing multiple texts")  
  
  
multiple\_clf = train\_and\_test\_model(multiple\_frequency\_vectors, multiple\_labels, print\_results=verbose)  
  
  
# get the shared quotes and the text they originate in  
shared\_info = get\_shared\_info(text\_ids,   
 text\_labels,   
 jpm\_intertext\_data,   
 limit=length\_limit)  
  
# filter down to just shared text and then create vectors  
shared\_text = [d[0] for d in shared\_info]  
shared\_frequencies = multiple\_vectorizer.transform(shared\_text)  
  
# make predictions annd get confidence measures of the decisions  
shared\_pred = multiple\_clf.predict(shared\_frequencies)  
confidence\_measures = multiple\_clf.decision\_function(shared\_frequencies)  
  
  
# create results dictionary  
results = {label:0 for label in text\_labels}  
  
# populate results dictionary  
for p in set(shared\_pred):  
 results[p] = list(shared\_pred).count(p)  
if verbose:  
 print(results)