Data Transformation, Metrics, and Smoothing: An Application of Topological Data Analysis

George Spahn   Shivam Nadimpalli

Term Paper for MATH1810A: Applied Algebraic Topology

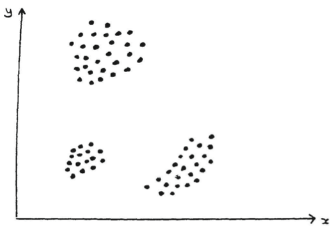
Abstract

The work described in this paper is a part of the subject of *Topological Data Analysis* (TDA), an area of research which has been developed over the past two decades. TDA allows for a framework to analyze point cloud data sets from a geometric point of view, which may reveal previously hidden, qualitative properties of data. Here, we describe an application of TDA to point cloud data derived from multiple corpora of English text documents. We attempt to cluster text documents from different corpora on the basis of varying parameters such as author, word usage, general sentiment, etc.

1 Introduction

Topology, put simply, is a branch of mathematics which studies notions of shape. More precisely, topology is the study of properties of space which are preserved under continuous deformation (e.g. stretching or bending), but not tearing or gluing. *Topological Data Analysis* (TDA) adapts the techniques for studying the properties of shape and applies them to suitably defined data.

Consider, for example, the following scatter-plot:



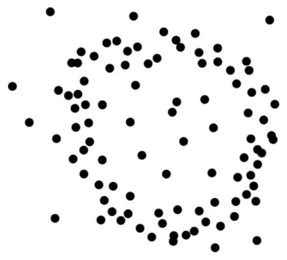
We can see that the above dataset can be clumped together into three disparate groups, each of which is a "connected components". *Connectedness* represents the simplest property of shape, and can thus be used to classify data into distinct categories. Various other properties, such as the number of *loops* or the number of *n-dimensional voids* in a data set, are other parameters which allow us to analyze a dataset using topological approaches.

In Big Data and Data Science problems, shape arises because of the fundamental notion of similarity and/or dissimilarity between the data points. Various metrics such as the Euclidean distance, correlations, weighted edges, etc. define the shape of the dataset, which can then be analyzed using TDA with no additional model information.

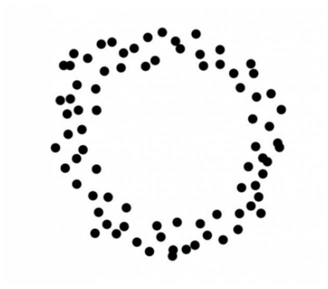
Most real-world datasets are incredibly large and dense, with multiple high-dimensional structures, loops, clusters, voids, and flares appearing within a single dataset, and this seemingly trivial concept of the study of "shape" can be exploited to solve complex problems with such datasets.

In order to be able to use topology-based approaches to analyze a dataset, it must be present in a form conducive to such analysis. In order to generate a *point cloud* from the input data, some form of *vectorization* is usually performed. This also equips the vectorized data space with a notion of distance. And in order to increase the resolution/clarity of results obtained using TDA, noise-reduction and data smoothing may be required.

For example, consider the following point cloud data:



While this may not seem to reveal much about the dataset, noise-reduction and smoothing allow us to clearly see the presence of a loop and a void within the data:



Recent applications of TDA have obtained remarkable results in fields ranging from chemistry and neuroscience to linguistics and music.

TDA, thus, not only provides new ways of data visualization and compression, but also allows for interactivity in data analysis, while offering greater control over scale and resolution.

In this paper, we present an application of TDA to multiple real world datasets: two corpora of English text documents obtained from Project Gutenberg, a digital library of free e-books, and a corpus derived from 1000 IMDb (Internet Movie Database) reviews. This project was based off of previous work done by Dr. Jennifer Kloke at Ayasdi – our aim was to recreate, analyze, and add to previously obtained results.

In spite of a very elementary approach to parsing the documents and generating a metric on this corpus, we were able to obtain interesting results using Ayasdi's *Mapper* algorithm. We also had a fairly simplistic approach to data-smoothing and noise reduction. More sophisticated parsing and smoothing methods could potentially yield better results.

We also include a discussion of previous work done by Dr. Jennifer Kloke at Ayasdi, a comparison with our results, and possible future directions.

2 Methods

In order to be able to run computational topology-based tools over a dataset, the input must be a set of data points equipped with some notion of distance, a dissimilarity function, or another metric. Therefore, our initial dataset, an "unstructured" corpus of $\sim$20,000 text documents, needs to be transformed to a form amenable to topological data analysis tools.

Various methods exists in order to *vectorize* a corpus, essentially generating an embedding of the words or documents in a vector space. GloVe, Word2vec, Latent Dirichlet Analysis, and Latent Semantic Analysis are some such methods. We use Latent Semantic Analysis (LSA), specifically the *term frequency—inverse document frequency* or *tf-idf* formulation, in order to generate vector spaces based on our corpora.

LSA offers several advantages over contemporary packages such as GloVe. Moreover, previous work done by Kloke et. al. was also based on LSA, allowing us to use their results for comparison.

However, the output of LSA is still "noisy": for example, certain words in the English language, such as 'of' or 'the', appear so frequently that they contribute very little to the "identity" of a particular document. *Smoothing* is the process of reducing the noise in our input, and our implementation used a simple but effective smoothing method.

Topological Data Analysis was performed using Ayasdi's core platform tools, central to which is the *Mapper* algorithm.

Here, we provide an in-depth summary of the tools and methods described above.

2.1 Latent Semantic Analysis

Latent Semantic Analysis, or *LSA* constructs a data matrix based on a provided corpus of documents. TDA requires some notion of distance, similarity, or dissimilarity between its input points, and LSA can be used to generate such a metric on text documents.

LSA, as we shall soon describe, is very simple in its approach to parsing text and generating a metric, but offers three important advantages over its modern counterparts (e.g. Word2vec, GloVe, etc.): speed, accessibility, and control. We were able to code out LSA from scratch (all our code is available here), and this allowed us to experiment with multiple LSA formulations so as to obtain the best results. LSA also offers greater control over the scale and resolution of the

TDA-generated results than modern tools like Word2vec.

The methodology behind LSA can be best understood with an example.

Consider the following corpus of two documents:

A: `‘‘Topologists like honey.’’` B: `‘‘Honey badgers do not care.’’`

Using Python's built in methods and modules (e.g. Re), it is possible to strip each document of punctuation, numbers, and special characters. We also convert all text to lower case – while this may seem disadvantageous, in reality, proper nouns contribute very little towards the "identity" of a document, and also get phased out in most parsing methods.

We thus obtain a "Bag of Words" (BoW) for each document in our corpus:

$\text{BoW}_A =$ `[‘topologists’, ‘like’, ‘honey’]`
$\text{BoW}_B =$ `[‘honey’, ‘badgers’, ‘do’, ‘not’, ‘care’]`

Note that words appear as many times in each BoW as they do in the corresponding document. For example, the BoW corresponding to "My name is Mr. Apple Apple." would be `[‘my’, ‘name’, ‘is’, ‘mr’, ‘apple’, ‘apple’]`

In order to get to "numbers" from these individual words (since our goal, after all, is to generate a metric over this space), we chose the simplest possible parameter: word frequencies. After generating a corpus-wide dictionary, we generate dictionaries for each BoW, with words as keys, and word frequencies and the values for the keys.

Thus, the corpus-wide dictionary is:
```
corpusDict = [‘topologists’, ‘like’, ‘honey’,‘fun’, ‘badgers’, ‘do’,
                            ‘care’, ‘not’]
```
And the frequency dictionaries for each BoW are:
```
corpusDict_A = [‘topologists’ : 1, ‘like’ : 1,
                  ...‘care’ : 0, ‘not’ : 0]
corpusDict_B = [‘topologists’ : 0, ‘like’ : 0,
                  ...‘care’ : 1, ‘not’ : 1]
```
Based on these word frequencies, it is possible to generate a word-matrix:

Table 1: Word Frequency Matrix

	topologists	honey	...	care
A	1	1	...	0
B	0	1	...	1

Thus, we have a matrix that can be inputted into TDA software; however, there are a few shortcomings with this expression, which is why we use the $tf - idf$ formulation of LSA.

2.2 The $tf - idf$ Formulation

The problem with the previously described counting strategy lies with the fact that we use a lot of words that contribute very little to the identity of a document. For example, the most commonly used word in the English language is the word 'the', which forms 7% of our daily vocabulary.

Proper nouns and pronouns also add to this "noise". Smoothing, or noise-reduction, is thus necessary in order to not only provide clearer results, but also to decrease the time required for computation.

Thus, instead of counting the number of occurrences of words in each document, we use the *term frequency–inverse document frequency* or $tf - idf$ score of a word in a document to measure its importance. The $tf - idf$ formulation of the word-matrix obtained as discussed in **2.1**.

We choose to not go into the technicalities of $tf - idf$ here, but all our code is available online on GitHub. There exist multiple formulations for the term frequency (tf), and we chose the one that allowed us to obtain the best results.

Continuing with the previous example, the $tf - idf$ formulation of Table 1 would be:

Table 2: $tf - idf$ Matrix

	topologists	honey	...	care
A	1	1	...	0
B	0	1	...	1

2.3 Obtaining Corpora

A script to download random selections of files from Project Gutenberg failed, due to limitations enforced by the website. We manually downloaded files from Project Gutenberg, and also found a dump of pre-classified IMDb movie reviews online.

2.4 Mapper

We used Ayasdi's Workbench platform, at the center of which is the Mapper algorithm. The technicalities behind the working of Mapper were covered in class as well as in the final presentations by our peers. We choose to devote more time to this (somewhat interesting) application of Mapper.

Mapper allows us to look at and interact with high-dimensional data, which was crucial for this project.

2.5 Implementation Details

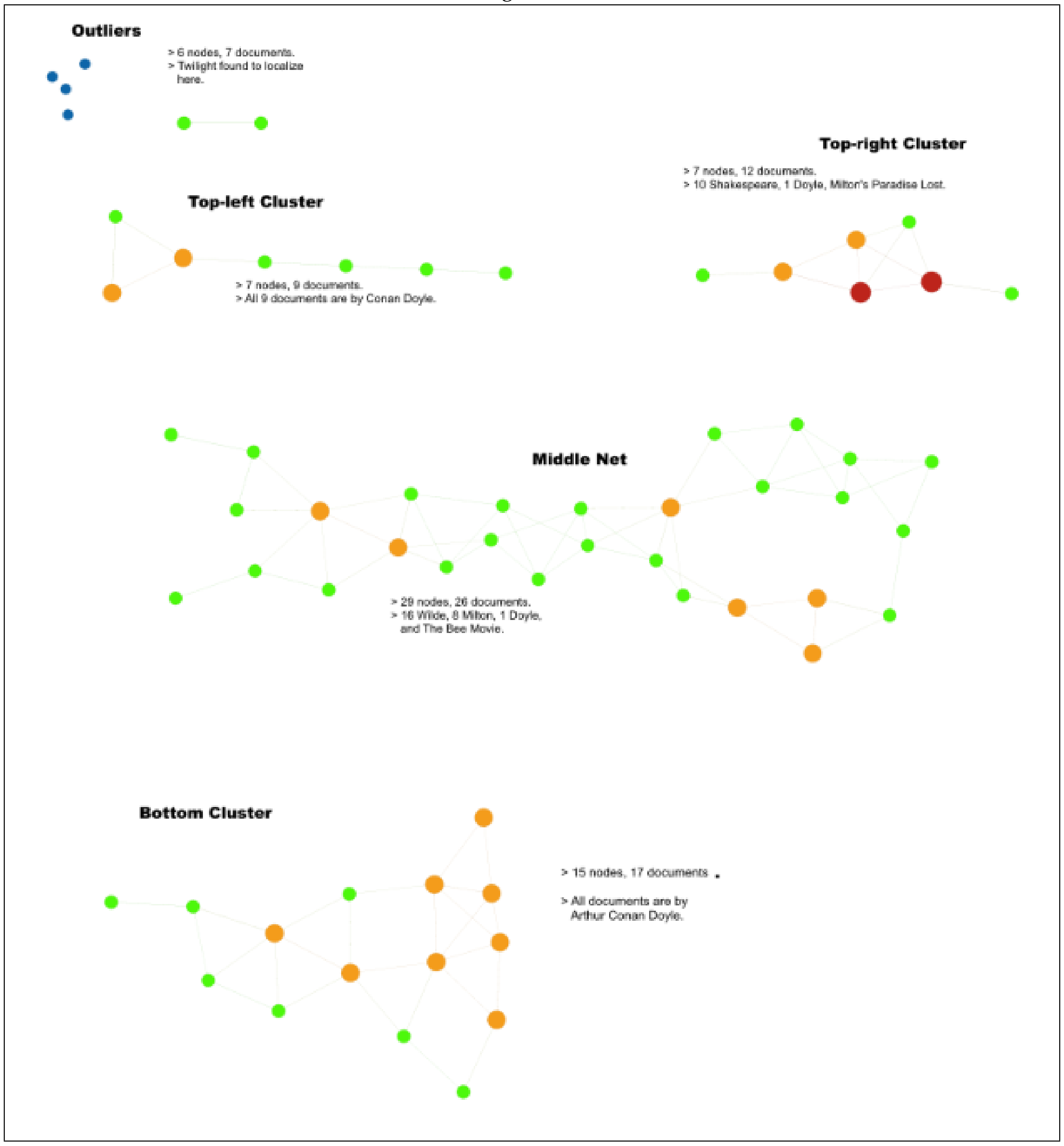
All code was implemented on our personal computers.

3 Results

3.1 Gutenberg I

We downloaded the majority of our text documents from Project Gutenberg, an online library containing over fifty thousand ebooks. We analyzed three main corpora for our project. The first included 10 works of Shakespeare, 17 by Oscar Wilde, 32 by Arthur Conan Doyle, and 10 by John Milton. Out of curiosity we also added Twilight, by Stephenie Meyer, and the script of the Bee Movie, for a total of 71 documents in the corpus. After uploading our TF-IDF matrix, we used to Mapper to produce the following model. It is based on a Hamming metric, which projects the data onto the 2-dimensional subspace spanned by the parameters which have the highest variance throughout the data.

Figure 3.1


Outliers
> 6 nodes, 7 documents.
> Twilight found to localize here.
Top-right Cluster
> 7 nodes, 12 documents.
> 10 Shakespeare, 1 Doyle, Milton's Paradise Lost.
Top-left Cluster
> 7 nodes, 9 documents.
> All 9 documents are by Conan Doyle.
Middle Net
> 29 nodes, 26 documents.
> 16 Wilde, 8 Milton, 1 Doyle, and The Bee Movie.
Bottom Cluster
> 15 nodes, 17 documents
> All documents are by Arthur Conan Doyle.

We see that many of the clusters consist primarily of documents by a single author. For example the bottom and top left clusters are entirely works by Doyle, and the top right cluster is over 80% Shakespeare. Paradise Lost by Milton was also clustered with Shakespeare, which makes sense because it is written in blank verse similar to the style of Shakespeare.

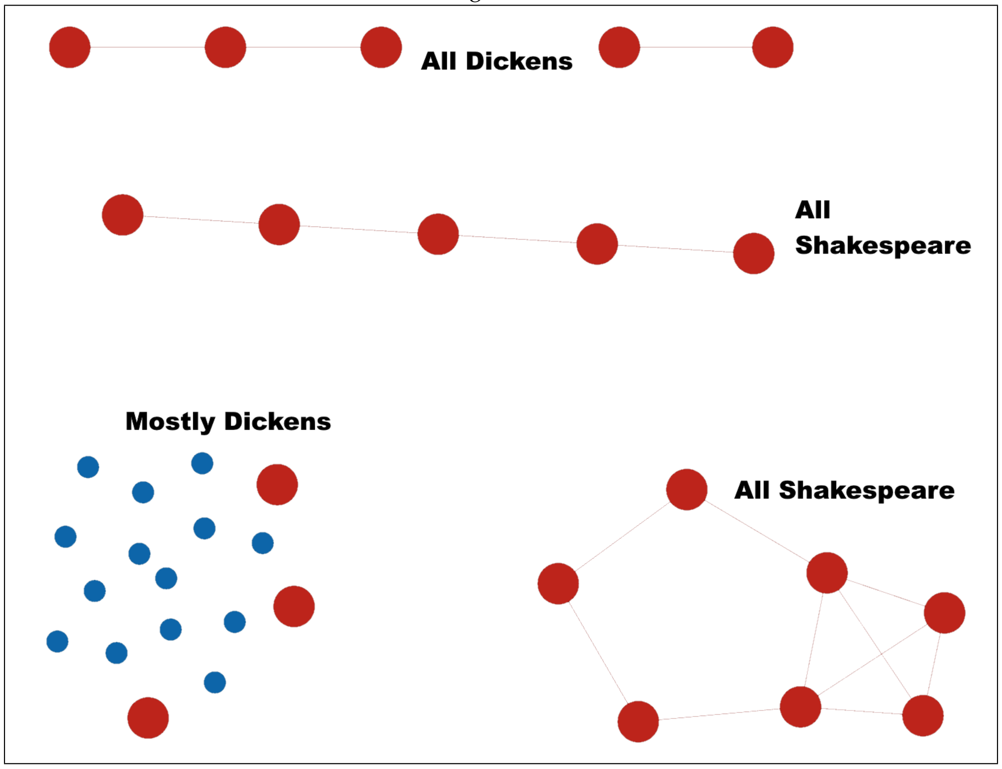
In the middle cluster we noticed that most of Miltons work is on the left, with Wildes work on the right, and the Bee Movie script on the far right. Also the upper Doyle cluster consists of all his short stories, where the lower cluster has his longer works.

Overall, we found the Mapper algorithm to be successful at distinguishing the works of different authors, based solely on our semantic analysis data.

3.2 Gutenberg II

For our second corpus we wanted to restrict our pool of authors, and attempt to distinguish the works of just two authors. We used 20 documents by Dickens, and 20 by Shakespeare. We hope to see two distinct clusters for each author. The following model uses Mappers categorical cosine metric, with the color based on a density lens function.

Figure 3.2



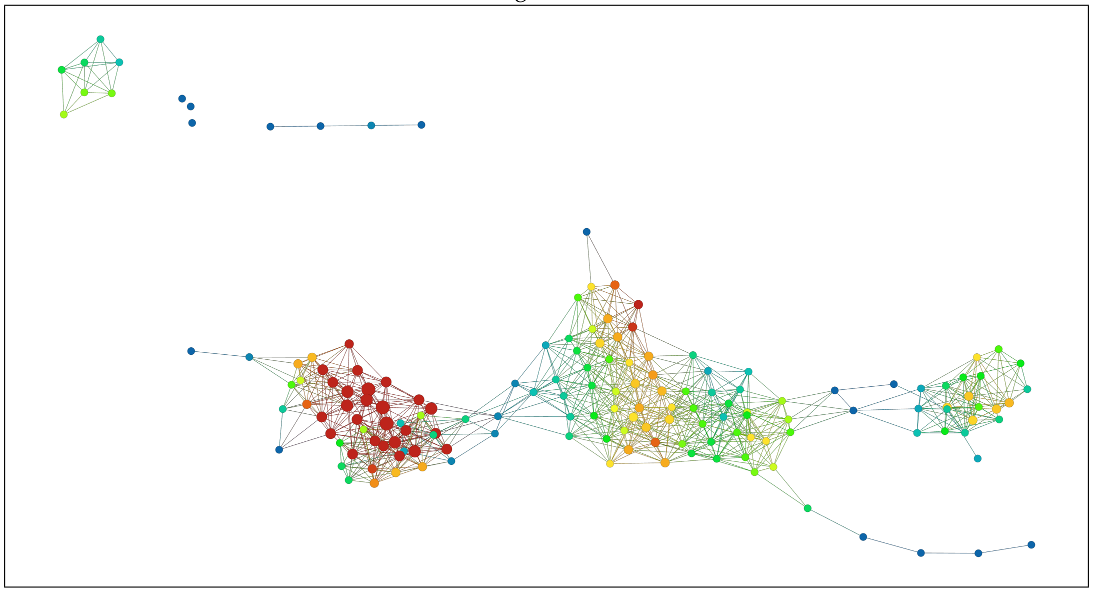
We were able to get clusters based on author, despite not having two distinct groups of points in the model.

3.3 IMDb Reviews

For our third corpus we found a dump of movie reviews, with each categorized as either positive or negative. An example positive review is “The songs were the best and the Muppets were so hilarious.”, and a sample negative review is “I wasn’t the least bit interested.”

We had a total of 500 positive reviews and 500 negative ones. We were hoping to use Mapper to distinguish between these two types of documents, or provide insight on how they were related to each other. After performing LSA, Mapper produced the following model:

Figure 3.3



While at first it seems like there are several clusters in the data, open closer examination the positive and negative reviews are dispersed randomly throughout each cluster. We could not finding any significance to the location or connections of the good/bad reviews.

4 Discussion

Our Results

LSA and Mapper provided successful results with the first two corpora – we believe this is most likely because of the distinctive writing styles and word usages of the chosen authors. However, we failed to obtain significant results with the IMDb corpus: We attribute this failure to extremely short length of the majority of the documents in the corpus.

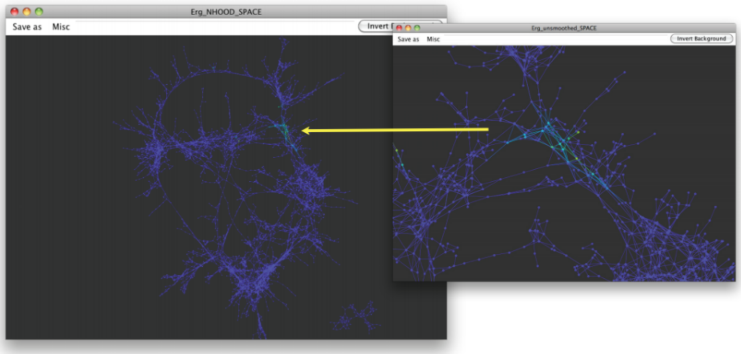
We propose that LSA, thus, is much more effective when each document contains many words, so the TF-IDF scores are more smooth. We think that for this type of problem, simpler clustering techniques such as just looking at the frequency of words like "good" and "great", would produce better results.

Comparison with Kloke et. al.

The paper used a $tf - idf$ formulation that was slightly different from the one we used in our project. They also calculated log-likelihood scores for the words in their corpus of documents, and smoothed out the words below a particular log-likelihood score threshold.

Access to greater computational power allowed Kloke et. al. to analyze a corpus of approximately 30,000 text documents from Project Gutenberg. Parsing 71 documents, on the other hand, took 17 seconds on our personal computers.

Figure 4.1: Ayasdi's *Gutenberg Space*

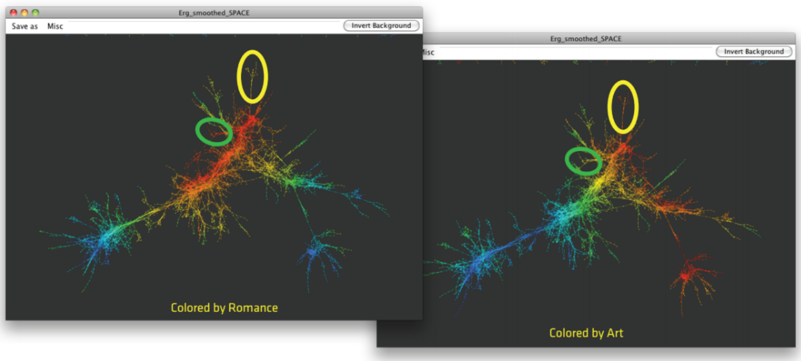


The paper used a graph theoretic approach to smoothing the data and removing unnecessary words (columns) from the $tf - idf$ matrix. The idea is to combine two columns if the words are very similar in meaning, or are different forms of the same word. So we can define a graph where all of the words in the corpus are vertices, and edges are created based on words that are similar. The paper defines the distance between a word and its plural to be 0.001, a word and its possessive to be 0.01, and a word to its lowercase version to be 0.05.

Additionally, the paper uses the online synonym dictionary "WordNet" which provides groups of synonyms, and also links groups that have the same meaning but at a different level of generality. For example the group for "furniture" would be linked to the one for "bed" which would be linked to "bunkbed". The paper assigns the distance between two words in the same synonym group to be 0.1, and the distance between words that are in linked groups to be 0.5. Finally, the distance between words with different endings including "-ing" and "-ness" were defined to be a distance of 0.6 apart.

We can now can use an algorithm to find the minimum distance between any two vertices in this graph, and use this as a metric between the words in the corpus. The paper then merges all words that are within a certain distance of each other and thus eliminates many unnecessary columns from $tf - idf$ matrix, allowing for more smooth data and faster computation times.

Figure 4.2: "Romance" and "Art" lenses



The paper also sought to analyze the corpus through lenses related to the themes present in the documents. To do this we define a list of words that is representative of the given theme. For example the paper defined a crime lens, in which the list of words consisted of mystery, murder, police, etc. For each document, the lens function is equal to the sum of all all the $tf - idf$ values of words in that list. We can then color our resulting models according to these lens functions to look for patterns and clusters in the data.

Limitations and Future Directions

We were mostly limited in our approach by our computational power.

Potential future approaches include alternative smoothing metrics, such as using a set stop-list to delete filler words and noise, and analyzing larger corpora of documents, and alternative word-vectorization tools such as Word2vec.

Acknowledgments

We would like to thank Dr. Sara K. Verovšek for her guidance on formulating this project, providing access to Ayasdi's Workbench, and for teaching an incredible MATH1810A this semester. We would also like to thank Dr. Jennifer Kloke of Ayasdi for guidance on text smoothing and transformation.