A Clustering Model for Ancient Chinese Poems

Jiaxin Guan
Graduate Student
jguan@princeton.edu

Abstract

In this project, we use two different clustering models, namely K-Means and DBSCAN to cluster a collection of Chinese poems from the Tang dynasty. In the feature extraction step, the body of each poem is represented by a bag-of-words representation where each word is a single Chinese character, followed by applying tf-idf and choosing the top features. Additionally, we create another seven features that captures the strains requirements for each poem. The clustering models are then trained on these features and we demonstrated how one can use the clustering to result to answer questions about Tang dynasty poems such as "which poet has the most consistent writing style?"

1 Motivation

Chinese ancient poetry, especially poems from the Tang dynasty, has been an important part of Chinese culture, as well as a masterpiece in ancient literature. These poems have been widely studied by scholars not only in China, but also worldwide. However, there is one important limitation of such literature studies: these studies are highly subjective. Quantitative analysis are seldom used in such fields; most of the decisions and deductions made are purely based on the researcher's own experience and judgment, and hence often times tend to be empirical and biased. In this project, we aim to use machine learning to build quantitative models for ancient Chinese poems, which while definitely not able to replace the role of researchers, can perhaps become tools that the researchers can use to aid their own research. For example, some questions that we hope to answer by building a quantitative model might include:

- Which poet has the most consistent writing style in his life?
- Which poets have similar styles in their writing?
- Is there a trend in the style of the poems that has developed over time?

2 Dataset

The dataset that we use is the Github "chinese-poetry" repository [1]. In particular, we take advantage of its collection of 57,613 poems from the Tang dynasty, located with the folder labeled "json" (it's a bad naming by the repo maintainers, but "meh").

The dataset is presented in JSON format, with 1,000 poems in each JSON file.

Each entry contains information about a single poem, which consists of the following fields:

- **strains**: The tone analysis of the poem. Presented as an array of strings, with each string corresponding to each verse in the poem. We will give more background and discuss more about this in section 4.2.
- **author**: The author of the poem.
- **paragraphs**: The body of the poem. Represented as an array of strings, with each string representing a single verse from the poem.

- **title**: The title of the poem.
- **tags**: Tags added by the curators of the repo depending on the poem's topic. However, only around 4,600 out of 57,613 poems have tags. Hence, we do *not* use the tags at all in our project.

3 Our Task

In this section we will clearly define our task for this project.

3.1 A Failed Attempt

Originally, we had a different goal for our project, which is what we wrote about in the project proposal. We eventually deviated from plan, but as we have spent a non-trivial amount of time working on that, we think it's reasonable for us to write about it briefly.

The motivation originates from the fact there is a non-negligible portion of poems that have unidentified authors. And a natural task for us to do is to use machine learning methods to help identify these unknown authors.

The method that we applied hoping to achieve that is to build a supervised classification model with the poet names as the classes. However, two issues occurred when we were implementing the project:

1. **The total number of poets is large**, which leads to a large number of classes. There are 3,719 different poets in total, and that would lead to 3,719 classes in our classification model.
2. **A significant portion of the poets have very few poems**. This means that we have very few datapoints for the classes that correspond to these poets. Hence the model is poorly trained on these classes.

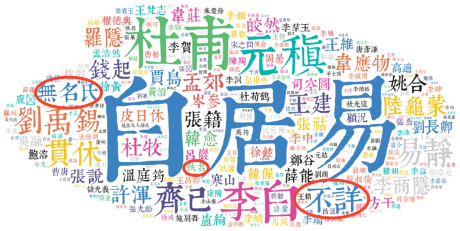


Figure 1: Word cloud for authors of poems from the Tang dynasty. The three circled authors are "noname", "anonymous", and "unknown", from left to right respectively.

Due to these two issues, our trained model has an accuracy of close to 0.

There's an obvious way to potentially fix these two issues altogether: only focus on the top k poets for some k. We deem this approach unreasonable for the following reason: the probability that a poem with unknown author is written by a well-known poet is close to zero. This is a simple result by the Bayes Theorem, as the poems of the well-known authors are well maintained, and hence the event that one of these poems has lost its author information in publication is unlikely. For example, the author with the largest number of poems is *Bai Juyi*, who is the huge blue name in the word map above. He has almost 3,000 poem creations (which is more than 5% of the total), whereas other famous poets only have at most several hundred poems. The reason that he has so many poems is not that he's way more productive than others, but because his family was in the publication industry. It's hard to imagine that one of his poems would lose its poet information in circulation.

Though a failed attempt, this model is still valuable to us, as the feature extraction techniques that we used are mostly reused in our new task, as defined below.

3.2 A New Task

Having observed the failure of our previous attempt, we decide to find a new task that can be reasonably achieved. Hence, our new task is to build a *clustering model* for the poems. This new task, compared to the previous attempt, has the following differences:

- The number of clusters need not to correspond to the number of poets, which ameliorates the issue of having too many classes.
- Since we the clusters no longer correspond to the poets, we have no known ground truth for the data. Hence, our model has to be *unsupervised*.

There are three important components in our project.

The first is feature extraction. Given the raw text of the poems, how should we represent it as machine conceivable features? What other auxiliary data about the poem can we use as the features? This is the part where most interesting stuffs happen. Notice that this step we can reuse from the previous failed attempt. Notice that We will discuss in details about this in section 4.

The next task is to train the actual clustering model. Which one(s) of the clustering models should we choose? How should we tune the parameters to use for each model? We will discuss this in section 5.

The remaining task is to evaluate our result. How good/bad does it perform? What problems can we solve with our model? We will also talk about our evaluation in section 5.

4 Feature Extraction

4.1 Paragraphs

The paragraphs of the poems are the most important data in the dataset and something that we definitely want to use in the training of our model. The paragraphs of a poem is a collection of the verses, represented by an array of strings int he dataset. A typical poem with two verses, *Jing Ye Si* by *Li Bai* looks like this:

床前明月光，疑是地上霜。

举头望明月，低头思故乡。

Each verse further consists of two halves, usually separated by comma and of the equal length. For example, for the first verse from above, its two halves correspondingly are "床前明月光" and "疑是地上霜".

A major challenge in clustering Chinese text is due to its linguistic differences from English.

Chinese has no inherent definition of "words". This posts a challenge on how we would represent the content of the poem. The common method of using "bag of words" representation doesn't directly work, as there is no simple way for us to define what a word is. To work around this, there are two popular methods used.

- To perform an initial segmentation module [3] that is intended to break a Chinese text into words. Most of these segmentation modules are dictionary-base [2] which uses a pre-determined dictionary to decide where to put delimiters between characters. The caveat is that all of these modules are designed for modern Chinese - there is no existing segmentation tools for old Chinese.
- To treat each separate character as a "word". This method is much more straightforward than the first one and has close to zero overhead. Notice that this method is possible because unlike English, Chinese characters bear meanings. Each character has meanings attached to it, and hence it's possible to use it as the unit for segmentation.

Both methods are widely used by machine learning models for Chinese language. To decide which approach to take, we notice that there are further differences between modern Chinese and old Chinese.

In old Chinese, especially in poems from the Tang dynasty, **the sentences and phrases are more compact**. For example, to express the same meaning of "to get close to something", modern Chinese would need two characters "靠近"(*Kao Jin*), whereas in old Chinese a single character "趋"(*Qu*) bears the same meaning.

In fact, most Tang dynasty poems are short. The majority of Tang has either 10 or 14 characters in each verse, and no more than 4 verses in total, leading to a total length of 56 characters. Given the limitation of its length, there is an important technique in ancient Chinese poetry called "炼字"(*Lian Zi*), which means "Character Curating" and corresponds to a process of choosing the best character to put in a specific spot. The motivation is that the poet should try to convey as much and as accurate meaning through each character as possible. Hence inherently, the characters in a Tang poem conveys way much information than they would normally do in modern Chinese.

Observing this fact, we deem it a better approach to use the second method, i.e., to treat each separate character as a word.

Using this technique, we are able to translate the paragraphs of the poems into a bag-of-words representation with a total of 9,526 words (features).

Two issues arise here.

1. The size of the dataset is too large to efficiently train on. We essentially have a matrix of dimension 57613×9526, which takes long to train even when we use a K-Means model with Mini-Batch.
2. A lot of the characters appear only a few times. As illustrated in figure 2, more than 50% of the characters only appear less than 20 times. These characters can hardly help us with the clustering, while largely wasting our computation power.

A common practice to deal with this is to use term frequency-inverse document frequency (tf-idf). The tf-idf value increase with the number of times that a term has appeared, and is offset by the number of documents that it has appeared in. tf-idf is the most common term weighting scheme in use today.

Eventually, we apply tf-idf to the paragraphs of the poems, and select the top 100 / 500 / 2000 features. (Notice that 2,000 is more than 20% of the total number of characters.) We will later train our model using these different numbers of features and decide which one is the optimal choice.

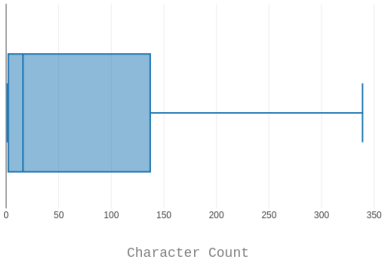


Figure 2: Box plot of number of occurrences for all the characters.

4.2 Strains

Strains are another important component of Tang dynasty poems. In Chinese, each character has a tone, which is either a *level* tone, or an *oblique* tone. In short, the strains of a poem pose a tone requirement on each of the characters used in the poem. The main purpose is to help with the musicality of the poem, so that the poem flows smoothly and is easy to read out aloud.

The strains of a poem is represented in a similar manner as the paragraphs. For example, the strains for *Jing Ye Si* is as follows:

平平仄仄平，平仄仄仄平。

仄平仄平仄，平平仄仄平。

We can see that it takes the same format, except now with different characters. Here the character "平" (*Ping*) stands for level tone, and "仄" (*Ze*) stands for oblique tone. Additionally, there is "○" that stands for a wildcard, i.e. either a level tone or an oblique tone can be used.

Obviously we should not simply use a bag-of-characters representation to represent the strains: there are only three possible character, and more impotantly, it is not what characters are in a poem that matters, but rather it is the ordering of these characters that matters. Therefore, after some extensive research on the strains requirements in Chinese poems, we use the following seven features to represent the strains for each poem.

1. **`ping_percentage, ze_percentage, wild_percentage`**: These three features simply calculate within the strains for a single poem, what is the proportion of the tones that are level, oblique, or wildcard, respectively. This is the most straightforward information that we can retrieve from the strains. It can help us answer the questions like "Is the poem mostly in level tones or oblique tones?". These features are represented as a float in $[0, 1]$.

2. **`ping_end_p, ze_end_p, wild_end_p`**: These three features examine the last strain of each verse. Like English poems, Chinese poems also pay attention to rhymes. It is usually required that the last characters of a poem are in rhyme. And hence, the strain requirements on these characters are extra important. Just like above, we calculate what portion of these requirements are level, oblique or wildcard, respectively. These features are represented as a float in $[0, 1]$.

3. **`symmetry_score`**: This feature calculates the level of symmetry between the strains the two halves of a verse. This feature originates from the fact that Chinese poetry pay close attention to symmetry between the two halves of a single verse. For example, if the first character of the first half is "up", then ideally in its corresponding position, i.e., the first character of the second half should be "down" for symmetry. Ideally, the strains should also follow this symmetry pattern. If a character is a level tone, then the character at the same position in the other half of the verse should be oblique tone. However, perfect symmetry is hard to achieve. Nevertheless the poets usually make their best effort. This feature calculates what proportion of the strains fulfills the symmetry requirement. For wildcards, we assume that they always fulfill the requirement. The feature is represented by a float in $[0, 1]$.

4.3 Others

Now we briefly discuss other information in the dataset and why we do *not* want to use them as features.

1. ***Title***. Unlike English poetry, the titles of Chinese poems often times do not tell about the content of the poem, but rather the background under which this poem was written. In fact, some author tend to use the titles almost like a preface to the poem itself. For exmaple, the poet *Bai Ju Yi* has a poem titled

   "予与故刑部李侍郎早结道友以药术为事与故京兆尹晚为诗侣有林泉之期周岁之间二君长逝李住曲江北元居升平西追感旧游因贻同志"

   (yes you read it right. It's 57 characters long) which roughly means

   I am an old friend with Mr. Li who served in the Criminal Justice Department. We shared a common interest in medicine studies. In my later years I become a friend with Mayor Yuan. We shared a common interest in poetry and we both wanted to pursue a recluse lifestyle. In the last year, I've learned that both of them has passed away. Li lived north of Qujiang, and Yuan lived west of Shengping. I visited these places in memory of my friends, and hence wrote this poem to people who we share common interests.

   Hence, we decide it's not appropriate for us to include the title in our features, as it does not give us any information about the style of the poem itself.

2. ***Author***. We decide to hide the names of the poets in order to see if just presented with the style of the writing, is the model able to show any latent structure based on the poets. Another reason, just as we have mentioned, the total number of poets (features) is too large for us to efficiently train a model using sklearn.

3. ***Tags***. The tags do contain valuable information about the topics of the poem. However, they are not created for every poem. Only around 4,600 out of 57,613 poems are tagged. Due to the large amount of missing data, we decide to not include tags as a feature.

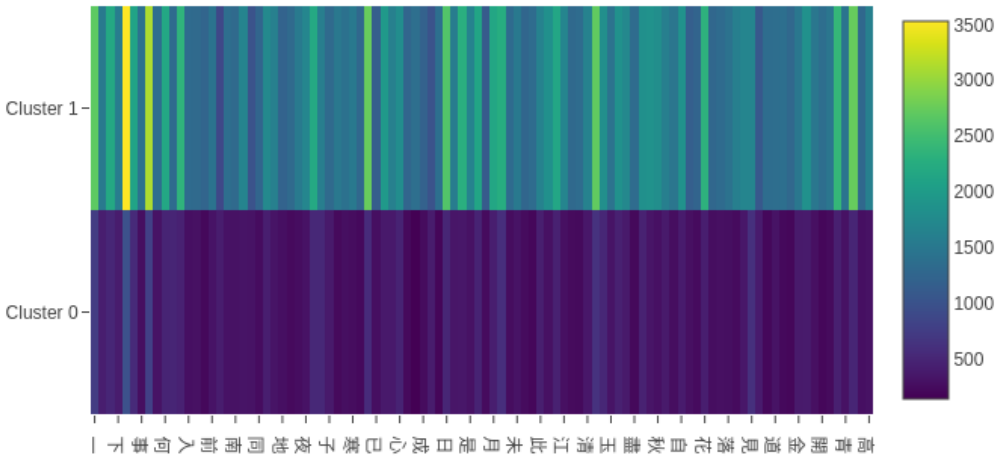


Figure 3: **Heatmap of the clustering result when applying K-Means to all the features.** The X axis are the features extracted by tf-idf, and the Y axis are the clustering results. The Z values (colors) represent the frequency that a specific word (feature) has appeared in responses in a specific cluster.

5 Model and Evaluation

Given the extracted features, i.e. a tf-idf representation of the paragraphs poems (for now, we will use the version with 2,000 features), and seven features from the strains, we proceed to train clustering models using these features.

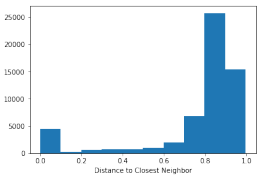
The list of clustering models the we have used includes:

1. K-Means
2. Spectral Clustering
3. Ward Hierarchical Clustering
4. DBSCAN

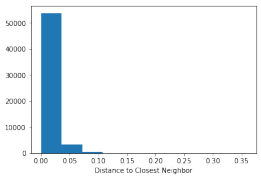
Among these models, we failed to train the Spectral Clustering and the Ward Hierarchical Clustering model on the data (every time we train them, the machine freezes and requires a hard reboot), likely due to the large number of samples and the limited computation power of my machine.

To start off, we combine the tf-idf matrix with the strains features to get one single feature matrix. Then we apply the K-Means model to it with the number of clusters varying from 2 to 128, evaluating the clustering result using the silhouette score. Intersetingly, we get the best result with the number of clusters 2. This is not in our expectation, as we conceive that usually the more clusters we have, the better clustering result we have.

Therefore, we decided to take a look at the heatmap of the characters corresponding to this clustering result, as shown in figure 3. We can tell from the figure that whether a character is in a poem does not affect the result of the clustering at all. We believe this is caused by the inclusion of the seven Strains Features. Therefore, we decided that we should apply a separate clustering model to the tf-idf matrix and the Strains Features.



a The tf-idf matrix



b Strains Features

Figure 4: Histograms for the distance to the closest neighbors in the tf-idf matrix and the Strains Features, respectively.

We first apply the K-Means model to both sets of features separately with a varying number of clusters. This time, for the tf-idf matrix, the optimal number of clusters is 16 with a silhouette score of 0.025, whileas for the Strains Features, we get an optimal number of clusters being 4 with a silhouette score of 0.62.

In terms of applying the DBSCAN model, in order to get meaningful clustering (i.e. number of clusters is greater than 1), we need that there is enough separation between the clusters. To determine if the data is suitable to apply a DBSCAN model on, we created histogram of each sample's distance to its closest neighbor, which will also help us in selecting an appropriate ϵ value for the DBSCAN model.

We can see from the figures that it is *not* ideal for us to apply a DBSCAN model to the tf-idf matrix, as almost all of the data points are at the maximum distance from its closest neighbor. If we apply a DBSCAN model to it, we will likely only receive one cluster. (And this is indeed what happens after we ran DBSCAN on this matrix). However, the histogram is more promising for the strains features. We can see that for a majority of datapoints, its distance to its closest neighbor is less than 0.07, and this is towards the minimum of such distances. Hence, we ran the DBSCAN model on the strains features, giving us a silhouette score of 0.70, which is indeed better than the K-Means model.

Recall that we created three different versions of the tf-idf matrix: each with 100, 500 and 2,000 features, and the version that we've been using is the one with $2,000$ features. We experimented with using a fewer number of features, and it turns out that by using just 100 features and K-Means, we get a silhouette score of 0.024, which is only slightly lower than the score of 0.025 when we use $2,000$ features. It also uses a total number of 16 clusters.

To convince ourself that we indeed get a better result by clustering the paragraphs and the strains separately, we take a look at the heatmap when we cluster only using the paragraphs, as shown in figure 5.

Gladly, we see that this heatmap does reflect that certain characters contribute greatly to a cluster. For example, take cluster 13, there are two characters that have really high weights, and they turns out to be "flower" and "spring". And hence we could conclude that this cluster corresponds to poems writing about springtime and that the model succeeds in clustering the poems into clusters with similar style and topic.

Now that given the clustering results, we can hopefully use them to answer certain questions about the Tang dynasty poems. Here as an illustration, we will try to use this clustering data to answer the question: which author has the most consistent style? To find this out, we first restrict our authors to authors who have at least 100 poems on record; otherwise, poets with only 1 poem are always the most consistent ones. Then, we group the rows by the authors and find the cluster where the majority of the authors' poems are in, and we divide the size of the cluster by the author's total number of poems to get a ratio between 0 and 1 as a consistency score. A consistency score of x means that an x portion of that author's poems are in the same cluster. The result is interesting. The top three consistent poets are 易静, 庞蕴, and 傅翕, each with a consistent score of 0.69, 0.61, and 0.56. That means for the most consistent author, 69% of his poems are in the same cluster. Interestingly, none of these are well-known poets. In fact, the first well-known poet comes in at number 18 on the list, which turns out to be 韩愈 with a score of 0.22, which is significantly lower than the top three. What does this indicate? It is a bit hard to say. Maybe that the more versatile a poet is, the more likely he is going to succeed? Or maybe well-known authors have more pressure from peers, so that there are

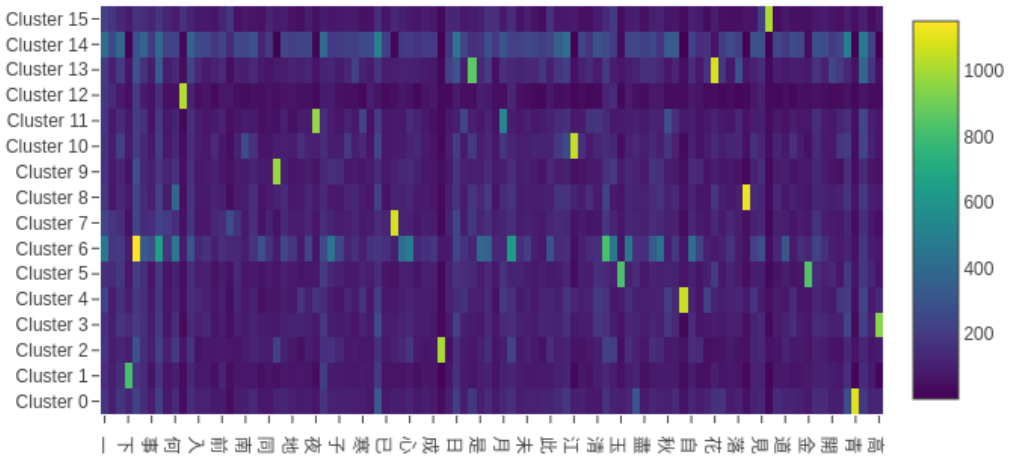


Figure 5: **Heatmap of the clustering result when applying K-Means to only the tf-idf matrix.** The X axis are the features extracted by tf-idf, and the Y axis are the clustering results. The Z values (colors) represent the frequency that a specific word (feature) has appeared in responses in a specific cluster.

certain styles and topics that they do not wish to touch? It is hard to come to a conclusion without extenside study of the field.

6 Conclusion

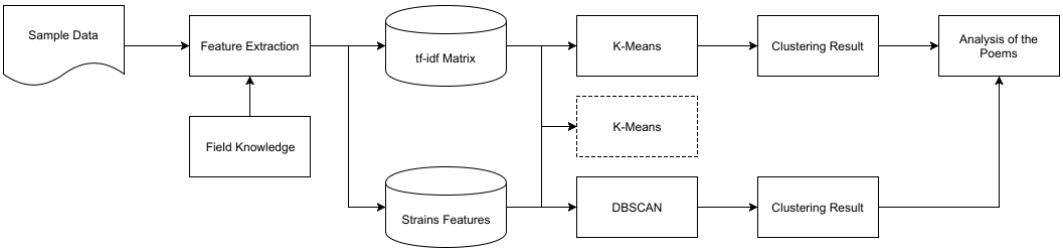


Figure 6: The flowchart of this project. The dashed box represented the clustering algorithm that we tried by did not work well.

To sum up, as shown in the flowchart in figure 6, we first apply different feature extraction techniques, most of which accustomed to the Chinese language. Then, we apply two different clustering models, namely, K-Means and DBSCAN, for the two different features. Lastly, we show how one can use the clustering results to answer certain questions about Tang dynasty poems.

References

[1] `https://github.com/chinese-poetry/chinese-poetry`.

[2] Ying Tan, Lan Huang, Hong Qi, and Yandong Zhai. Design and implementation of chinese text clustering system. In *2009 Fifth International Joint Conference on INC, IMS and IDC*, pages 1136–1140. IEEE, 2009.

[3] Taras Zagibalov and John Carroll. Automatic seed word selection for unsupervised sentiment classification of chinese text. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 1073–1080. Association for Computational Linguistics, 2008.