Nâzım: Critical Edition Tool

In this project, my aim is to build a critical edition tool for editors. Building a critical edition requires several steps, such as data collection, collation (aligning similar lines between editions), analysis on the differences between editions, and finally combining all editions into one file which what we call critical edition. In the project, I will try to build a collation and critical edition tool that can help editors. The tool is called as Nâzım which means editorHowever, the tool will not be a self-sufficient model, rather ease the workload of editors. The tool, as an experiment, is deployed into 16th century Turkish text and gives good result.

Research questions
The project has several research questions.

1. What are the methods for handling complex textual variation in historical poetic texts in manuscripts?
2. How does a poem change over time in manuscripts?
3. How can the process of making a critical edition be automatized as much as possible?
4. To what extent should we have the computer do the work to get a reliable result? Should we leave all the floor to it, or should we check every output it creates?

Background

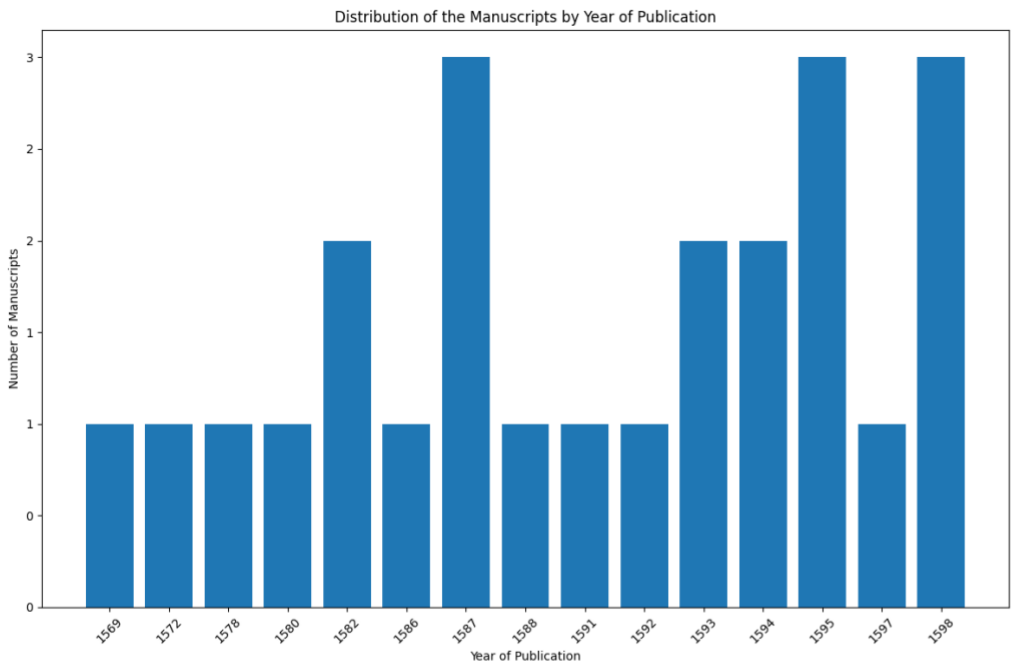
Analyzing manuscripts:
The method of manuscript comparison by aligning lines has been used before. Pire (2022) analyzed various manuscripts from Bâkî, a 16th-century Ottoman Turkish poet, by manually aligning the lines in 25 different manuscripts. Then, she explained how the lines in manuscripts changed over time. However, this work involved manually aligning similar lines, which can be quite labor-intensive for a total of more than 30,000 lines. The current work can be appreciated as an automated version of what Pire did since both uses the same data and aim to reach the same output.

Tools:

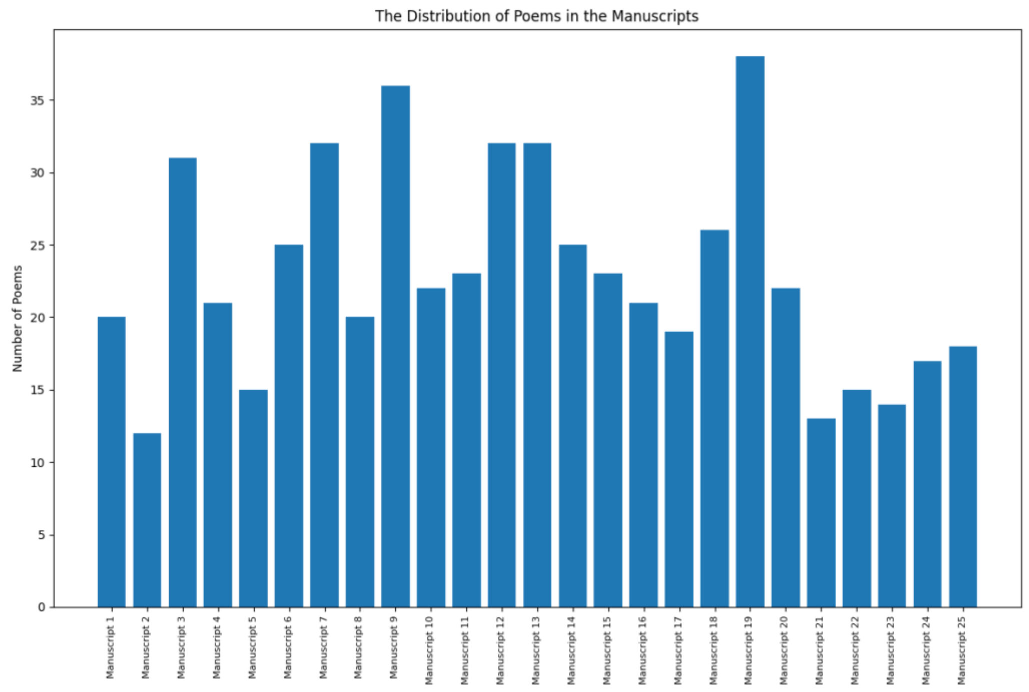
Computational tools have been deployed in the field of critical edition to help editors when they deal with multiple editions. Some of them have focused on both collation, in other words aligning similar lines or passages among different editions, and a critical edition preparation tool ("Classical Text Editor", 2023). Moreover, there were some trials on analysis of the differences between editions for the similar passages such as sentence graphs (Andrews, 2019). However, none of them considered Ottoman Turkish's features when it comes to build a critical edition tool. Therefore, certain aspects of the current models do not align with the needs of a critical edition tool for Ottoman Turkish. In this work, I will use the current methods used for text reuse to find similar lines among different manuscripts and deploy it for Ottoman Turkish texts.

Data:

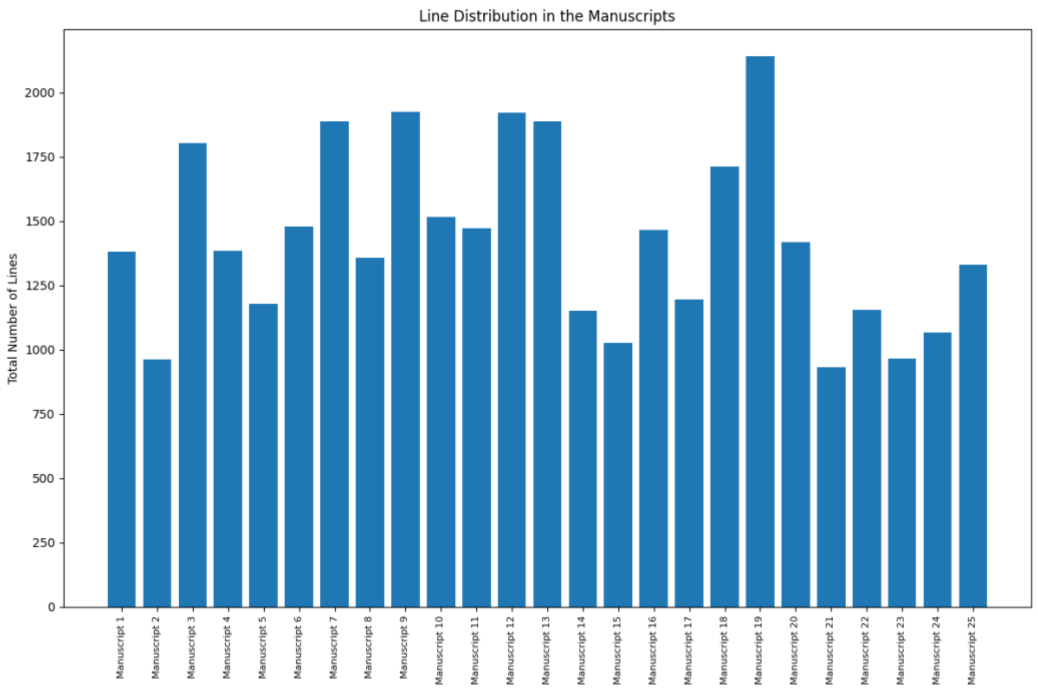
The data comprises 25 historical manuscripts consisting of poems. They have just one author, *Bâkî* (which means the Enduring). He was born in 1526 and died in 1600. Because of his fame and success in writing poems admired by the Sultan, he was entitled as “Sultan of Poets”. All manuscripts, in total, have 572 poems. The timeframe for the data is 16th century.



The earliest manuscript is from 1569 and the latest is from 1598. However, for some years, there are more than one manuscript.



The poem count per manuscript is between about ten and forty. Since new poems by Bâkî were published posthumously, the later manuscripts also include these new compositions. However, there is no strict linear increase in the volume of manuscripts over time.



The manuscripts have around 1,000-2,000 lines in general. Therefore, by examining the chart above, one might surmise that some manuscripts likely contain missing lines and others have additional lines.

However, after an elaborate analysis, it is observed that most of the lines are the different versions of the same poem. As an example, the lines below were taken from different poems in different manuscripts; however, as one can observe, all lines indeed represent the same content, albeit with slightly different wording.

In the year 1569: *eṭrāfa ṣaldı şaʿşaʿasın gūşe gūşe dür* (The **pearl** shed **its** splendidness everywhere)
In 1572: *eṭrāfa ṣaldı şaʿşaʿasın gūşe gūşe mihr* (The **sun** shed **its** splendidness everywhere)
In 1587: *eṭrāfa ṣaldı şaʿşaʿayı gūşe gūşe mihr* (The **sun** shed **the** splendidness everywhere)
In 1591: *eṭrāfa ṣaldı şaʿşaʿayı gūşe gūşe mihr* (The **sun** shed **the** splendidness everywhere)

Data Format:

The whole data is stored in one big folder. The folder has 25 subfolders which correspond to manuscripts. Each subfolder is titled with the name of the manuscript, for example, *1598 Tarihli İÜ Nüshas*ı (Manuscript of Istanbul University Dated 1598). Each subfolder has poem files in json format. Each of the json file's name shows the manuscript name, the genre of the poem, poem ID, and the first line of the poem. To exemplify, the manuscript name (1_iü_t_5523), the genre of the poem (kaside), page numbers of the poem (1b-3a), and the first line of the poem (biḥamdi 'llāh şeref buldı yine mülk-i süleymānī) for the json file "1_iü_t_5523_kaside_1b-3a_biḥamdi 'llāh şeref buldı yine mülk-i süleymānī.json". This information in the file name will be used later for the output file, critical edition.

The data is manually digitalized by domain experts and digitalized with a special focus on preserving orthographic features in the text. Thus, if there is a spelling error in the original text, the research data preserves such mistakes too.

I have done no cleaning at all since the data is well-formatted.

Data Processing:

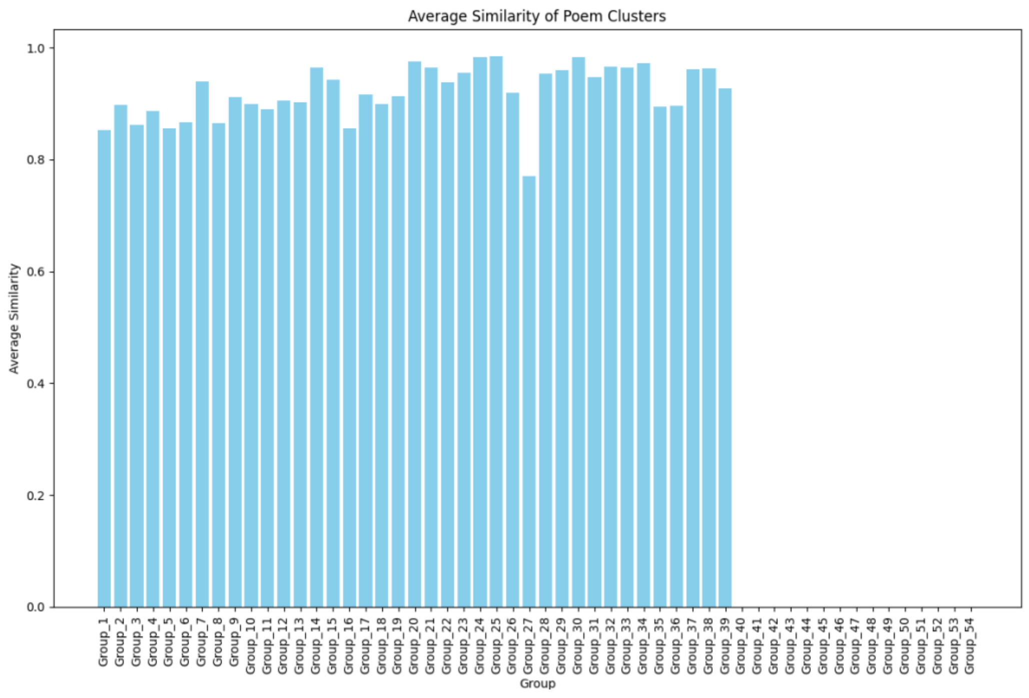
The data is then processed using three different Python files.

1. SimilarPoemsClusterer.py
    a. The script aims to cluster poems. To do this, it uses both sklearn library for feature extraction by using TfidfVectorizer and cosine similarity to cluster similar poems by using a similarity threshold. Firstly, the poems are loaded from json files and then by using TfidfVectorizer, the code converts each poem into a matrix then cluster each matrix via using 0.65 similarity threshold of cosine similarity. After testing various threshold values, such as 0.6 or 0.7, I determined that the optimal threshold is 0.65, as it can capture even highly deviated poems. To exemplify, a cluster has these two poems with the similarity value 0.95 and 0.68. While they have some same lines such as:

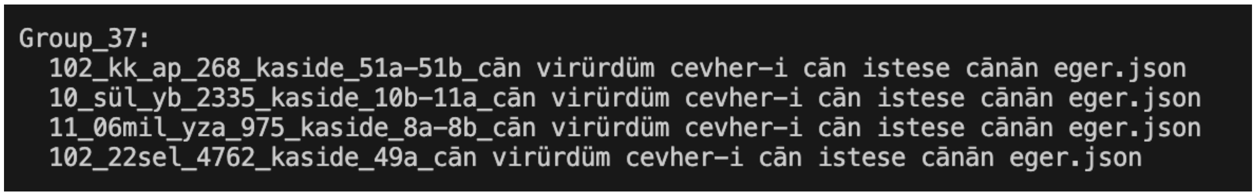
itdi şehri şeref-i maḳdem-i sulṭān-ı cihān / reşk-i bāġ-ı irem ü ġayret-i gülzār-ı cinān
they have also some lines that cannot be seen in both manuscripts such as the line below where is only in the poem with 0.68 similarity score in the cluster: *ne revādur fużalā ḳala ḳıbāb altında","kim görüpdür ola deryāyı ḫabāb içre nihān*

To capture such poems, I scaled the threshold to 0.65. Then the poems are clustered into subfolders in a folder called as “ClusteredPoems”. Furthermore, it is important to note that not clustering two poems in the same manuscript is highly considered, since it is not expected to see a poem’s different versions in the same manuscript but the different manuscripts.

Moreover, poems that do not share sufficient similarity with others are clustered individually.
As a result, the dataset consisting of 572 poems from 25 manuscripts was clustered into 54 different clusters based on the similarity threshold 0.65.



The chart above displays the average similarity for all poem clusters. While the poems in most of the clusters have high similarity embodied in the average for each cluster, cluster 27 has about 0.7 average similarity due to high variance between poems in the cluster. Since the last 15 clusters have just only one poem because they didn't exceed the similarity threshold 0.65 with other poems, there is a blank part in the chart. This step results in a folder where the poems are clustered into different subfolders. Each subfolder is seen as below:



```
Group_37:
  102_kk_ap_268_kaside_51a-51b_cān virürdüm cevher-i cān istese cānān eger.json
  10_sül_yb_2335_kaside_10b-11a_cān virürdüm cevher-i cān istese cānān eger.json
  11_06mil_yza_975_kaside_8a-8b_cān virürdüm cevher-i cān istese cānān eger.json
  102_22sel_4762_kaside_49a_cān virürdüm cevher-i cān istese cānān eger.json
```

As one can see, they start with the same line and if we look further, they are very similar to each other indeed, with some slight changes. After poems are clustered, the work continues with LineAligner.py file.

2. LineAligner.py
    a. This code basically aligns the lines in each cluster folder. It firstly converts each line into vector by using the bigram feature as Janicki applied in his article (Janicki, Kallio, & Sarv, 2023). Then align the vectorized lines with each other by using a weighted edit distance function, in other words, Wagner-Fischer algorithm (Wagner & Fischer, 1974). Then the code writes the aligned lines in each poem cluster folder into a different csv file.
3. CriticalEditor.py

a. The file aims to convert the csv files with aligned lines into docx files where the differences in each row are explained like in a critical edition. To do this, the file has some rules.
    i. If in a row, all entries are the same and they are not blank.
    ii. If entries in a row are different
        1. Find the most repeated line variant (excluding the blank one(s)) and add it to the document firstly. And then insert which edition include it into parenthesis just below the line.
        2. For the other variants except the blank ones, on the next consecutive lines, add them into parenthesis which also include which editions it can be encountered.
        3. For blank ones, state that this line is missing in the corresponding editions in parenthesis.

b. Thus, the only poem line which is not inside parenthesis is the most repeated line between the editions.

Reproducibility:

Reproducibility can be realized in various ways. The data in subfolders can be analyzed for language changes during the time or any further examination. Then each subfolder with a cluster is processed and turned into a csv file where the aligned lines can be seen in each row. Furthermore, the model can be employed to build a critical edition not only for Bâkî's texts but also for those of other Turkish poets. Moreover, if the data is competent for bigram analysis for vectorization part, the model can be applied to poems in other languages as well. The whole code is specifically designed for using it with data from other poets. The similarity threshold value for clustering the poems from the manuscripts can be changed based on the features of the data. The hyperparameters in LineAligner.py file can be changed based on the one's specific data. Without any further change, CriticalEditor.py file can be used for any aligned lines of poems stored in a csv file.
As Aksoyak showed that some of the manuscripts have poems that don't belong to Bâkî indeed (Aksoyak, 2005, 79. Although this study does not focus on identifying spurious poems in the data, the method can be adapted to detect potential forgeries by aligning Bâkî's lines with those of other poets, functioning similarly to a plagiarism detection tool.
Moreover, as the chart titled 'The Distribution of Poems in the Manuscripts' shows, not all manuscripts contain every poem; most include only a selection. In this case, in the future, the removed lines into the poem can be traced of.
What does the analysis show, how does it answer the humanities/social science research question? How do the results relate to possible prior and related work?

Critically analyzing the whole pipeline for potential bias and problems:

The threshold for tf-idf directly determines how the critical edition will look like. The ideal way of clustering would be manual poem clustering for such as small-scale projects.

Also, if two consecutive lines written in contrary ways in two different poems, the aligning code might not capture all similar lines for some situations such as:

Poem 1	Poem 2
A	A
B	C
C	B

Line structure is converted to:

A	A
-	C
B	B
C	

Thus, the aligning might not work in this example and the model cannot show the similar lines into the ciritical edition docx files.

Furthermore, the final output, which is critical edition docx file, by no means, should not be appreciated as a final critical edition but rather should be considered as an outline for a critical edition which editors can facilitate while preparing their critical edition. Sometimes, editors may want to ignore typographical variations across the editions. Moreover, tf-idf based clustering code can mistakenly cluster two random poem if they randomly share some lines; however, such a coincidence is not enough to align lines between these two poems. Therefore, manual checks on both the clusters and the critical edition docx file are the optimal way for a robust critical edition.

BIBLIOGRAPHY:

Andrews, T. L. (2019). Critical Edition as Process: A Digital Model PowerPoint slides. https://stemmaweb.net/wp-content/uploads/2020/09/ests_2019_presentation.pdf.

Aksoyak, İsmail Hakkı. (2005). Gelibolulu Mustafa Ali ve Baki'nin Münasebetleri [Künhü'l-Ahbar ve Divanlarına Göre]. In H. İnalcik, İ. E. Erünsal, H. W. Lowry, F. Emecen, & K. Kreiser (Eds.), The Journal of Ottoman Studies, XXV, Istanbul.

Classical Text Editor. (2023 December). Retrieved 20 December 2023 from https://cte.oeaw.ac.at/?id0=main.

Janicki, Maciej & Kati Kallio & Mari Sarv 2023: Exploring Finnic written oral folk poetry through string similarity. *Digital Scholarship in the Humanities 38 (1)*: 180–194. https://doi.org/10.1093/llc/fqac034.

Pire, Ş. (2022). *Yazmalarda Gizlenen Hikâyeler: Bâkî Divanı Yazmaları Üzerinden Şairin ve Şiirinin Değişimi* Unpublished master's thesis. Boğaziçi University.

Wagner, R. A., & Fischer, M. J. (1974). The string-to-string correction problem. *Journal of the ACM*, *21(1)*, 168-173. https://dl.acm.org/doi/pdf/10.1145/321796.321811.