Lab 3 – Parallelizing k-means

STAT 215A, Fall 2026

1 Introduction

Clustering can help describe patterns in dialects, but the number of chosen clusters can change how much detail is retained. In this report, the subsampling method of Ben-Hur et al. [1] is used to ask which k gives similar clusters when the survey respondents change. The goal is to find the most stable clusters of response patterns.

The supplied `lingBinary.Rdata` contains 45,152 respondents and 468 features. The six location and ID columns are excluded from clustering. The features contain no missing values, and all respondents are retained. The features are used without scaling or dimension reduction.

2 Similarity implementation

Pairwise Jaccard similarity was chosen as the similarity measure. It compares pairs of respondents grouped together in at least one clustering run and ignores pairs that are separated in both runs.

After aligning the labels of the q shared respondents, the pairs grouped together in the first clustering (A_1), second clustering (A_2), and both clusters (I) are counted. The Jaccard similarity is defined by

$$J = \frac{I}{A_1 + A_2 - I}.$$

The denominator counts pairs grouped together in at least one clustering run. Subtracting I prevents pairs shared by both clustering runs from being counted twice.

The number of distinct pairs of respondents in a group is calculated by $c(c-1)/2$ and is applied to each clusterings cluster sizes to obtain A_1 and A_2. For I, the respondents are grouped by their combination of labels from the two clusterings and then $c(c-1)/2$ is calculated again. When neither clustering groups any pair together, the function returns 1.

The similarity function makes one pass through the membership vectors to count cluster sizes and observed combinations of labels in dictionaries, then sums their pair counts. It does not construct the $q \times q$ pair matrix. Each dictionary contains at most q entries, so the function uses expected $O(q)$ time and $O(q)$ extra space. The expected time bound assumes that dictionary operations take constant time on average.

Numba is used to compile the loops. The provided benchmark, averaged over 100 calls took 0.412 ms at $q = 5{,}000$ and 1.990 ms at $q = 29{,}000$. A 5.8 times increase in input length produced about a 4.8 times increase in time, which is consistent with linear scaling.

Table 1 provides additional time measurements. Runtime grows approximately in proportion to the number of shared rows. Larger sampling fractions therefore increase the work even though the similarity function is linear for q.

Sampling fraction m	Rows per subsample	Shared rows q	Time per call (ms)
0.2	9,030	1,771	0.133
0.4	18,060	7,163	0.493
0.6	27,091	16,258	1.164
0.8	36,121	28,854	2.019

Table 1: Local similarity timings on aligned labels from two $k = 10$ fits to the linguistic data at each m.

3 Algorithm 1 implementation

For each $k = 2, \ldots, 10$, $B = 100$ independently drawn subsample pairs are compared. Each subsample contains 80% of respondents sampled without replacement. The two subsamples are clustered separately, and their membership vectors are aligned using shared row positions before calculating similarity.

In the algorithm implementation, scikit-learn's `KMeans` is used, with k-means++ initialization and ten initializations per fit to reduce sensitivity to poor initial cluster centers. The rest of the settings are default, including Lloyd's algorithm, a limit of 300 iterations, and tolerance 10^{-4}. Fixed seeds are used to make the sampling and fitting reproducible, different seeds are used for the two fits. There are 1,800 fits in total, each with ten initializations.

The inner repetition loop is parallelized with joblib's `loky` backend. This results in 100 tasks per k. Parallelizing only across k would instead create nine longer running jobs, with runtimes that vary across k.

The Slurm script requests twelve CPUs limited to one thread each, on one node in the `high` partition, and Python uses twelve worker processes. twelve CPUs per task was chosen to match the node's physical core count. The SCF run completed successfully in 1 hour, 37 minutes, and 34 seconds. A previous run with eight CPUs took 2 hours, 18 minutes, and 25 seconds, so the twelve-worker run had approximately 29.5% less elapsed time.

The experiment runs in a separate Python script and saves the 100×9 score matrix to `results/S.csv`. A plotting script then reads `results/S.csv` to generate the figures.

4 Result

Figures 1 and 2 recreate the histogram and empirical cumulative distribution plots in Figure 3 of Ben-Hur et al. [1], although in this report, Jaccard similarities are used instead of correlation similarities.

I would choose $k = 3$ as the cluster count. Its mean similarity is 0.972, its standard deviation is 0.018, and all scores are above 0.91. At $k = 4$, the mean falls to 0.886 and the similarity score has a minimum of 0.477, while only 47% of its scores reach 0.9. This supports stopping at three groups before the first substantial loss of stability.

For $k = 2$, the histogram has two peaks, one near 0.60 and another near 0.99. Although the median similarity 0.973, the lower scores show that some pairs of runs produce very different clusters. This suggests that the division into two groups is not consistently stable. Stability also generally decreases beyond $k = 3$. It improves somewhat at $k = 6$, but its mean similarity of 0.822 is still lower than at $k = 3$.

I consider this substantial evidence that three groups give a reproducible summary under the chosen inputs and algorithm. Altough several limits remain, such as The similarity scores reflecting both sampling changes and K-means starting points, and Large groups contributing many pairs and therefore hiding instability in small groups.

An alternative to choosing k based on stability, is the gap statistic, also discussed by Ben-Hur et al. It compares how tightly observations are grouped with what would be expected under a reference distribution without cluster structure.

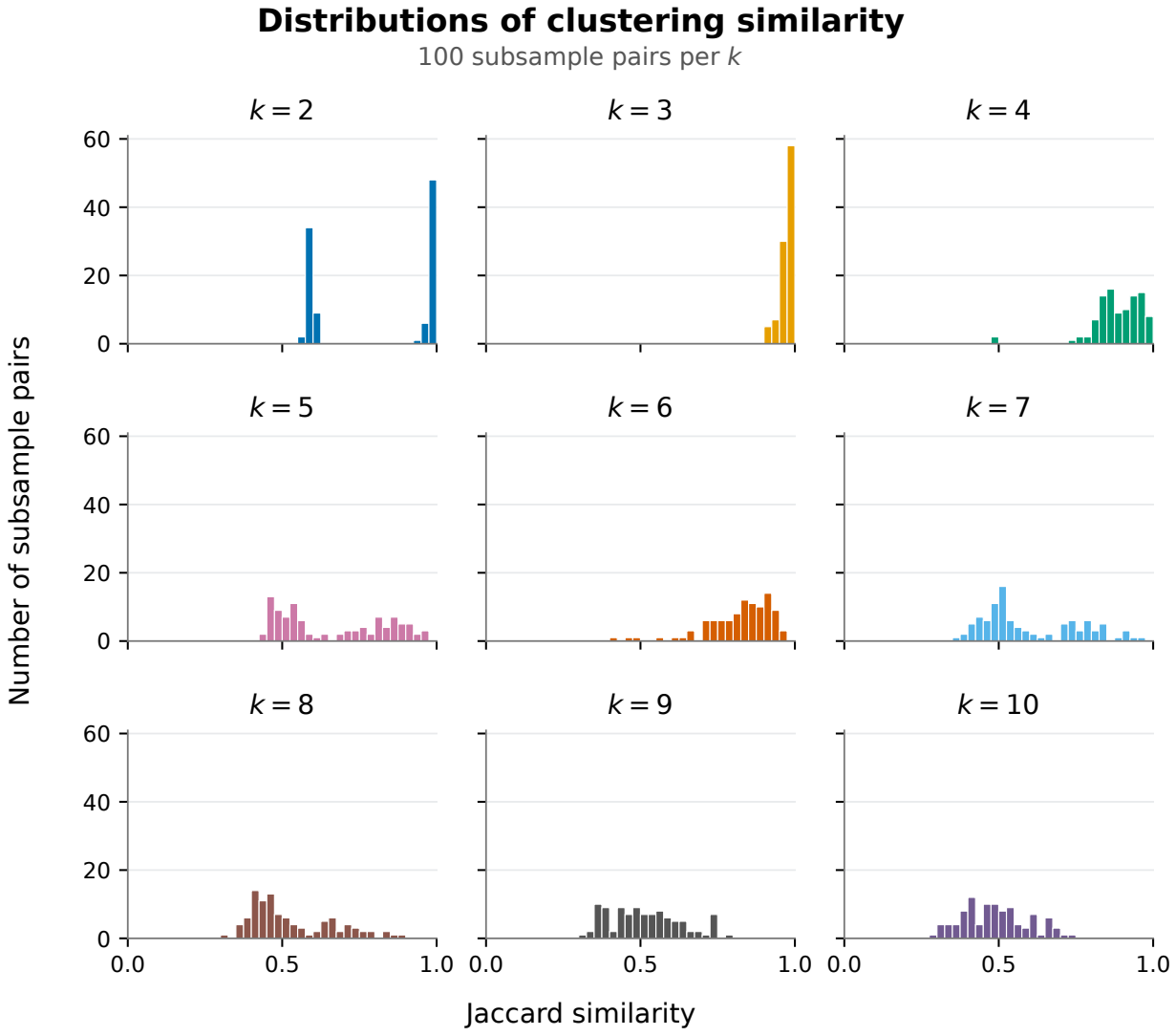


Figure 1: Similarity distributions for 100 independently sampled pairs per k, with $m = 0.8$.

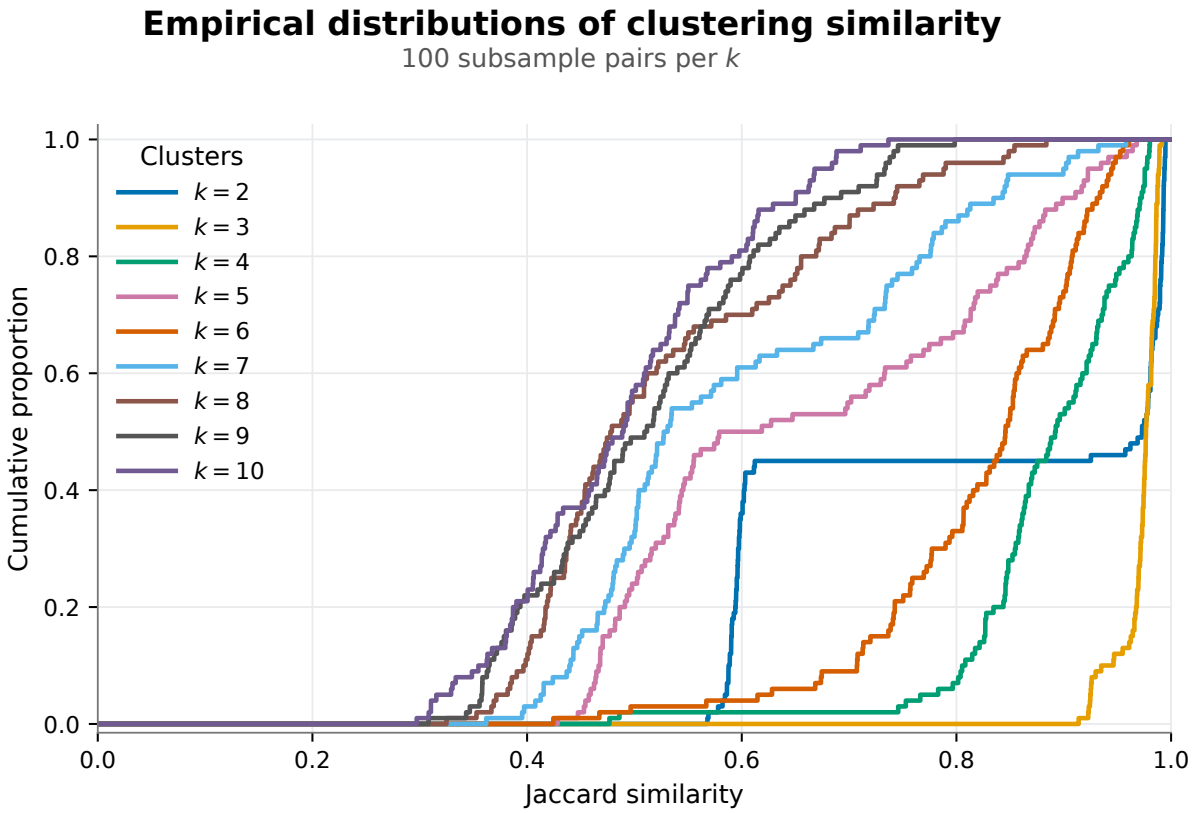


Figure 2: Empirical cumulative distributions of the similarity scores.

5 Academic Honesty

5.1 Statement

Dear Bin,

Academic honesty matters to me because dishonest or misrepresented work makes academic communication and scientific progress less efficient. I want to contribute to making the world better, so I feel that it is my responsibility to be academically honest.

5.2 LLM Usage

Coding

Codex was used to:

- Explain combinatorial pair counting.
- Explain how to compute Jaccard similarity without constructing the full pair matrix.
- Review my implementations, identify errors in my code, and check correctness.
- Check if benchmark timings were consistent with the expected time complexity, to ensure correct implementation.
- Provide Numba integration guidance
- Directly fixed wrong dictionary type declarations for Numba.
- Showed how to implement joblib parallelization and reviewed sampling, label alignment, random seeds, worker settings, and SCF integration.
- Generated docstring and comments in the code.

Overall it was a good experience, and saved me time, particularly in needing to look up documentation to understand a new implementation. It also helped me ensure that I was implementing things correctly and that I was on track throughout, which was very useful. I have used LLMs in the past, and wish to continue using them in the future for similar projects.

A link to the Codex chat can be found here.

Writing

- Helped interpret the lab instructions and organize required work.
- Reviewed draft implementation sections against code.
- Suggested wording and grammar corrections.
- Helped summarize LLM usage for LLM Usage section

5.3 Collaborators

I did not work with any human collaborators.

Bibliography

[1] A. Ben-Hur, A. Elisseeff, and I. Guyon. A stability based method for discovering structure in clustered data. *Pacific Symposium on Biocomputing*, 7, 6–17, 2002. `https://psb.stanford.edu/psb-online/proceedings/psb02/benhur.pdf`.