UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Unsupervised and Reinforcement Learning

Semi-supervised Clustering by Seeding

Written by:
Shivani Patel

May 16, 2022

Table of contents

Overview

The objective of this project is to implement the 2 semi-supervised variations of the K-means clustering algorithms, namely, Seeded K-Means and Constrained K-means[1], as illustrated in the paper Semi-supervised Clustering by Seeding. This report covers the custom implementation of the same, its comparison with similar other existing algorithms, impact of the hyperparameters (noise level in labelling, fraction of labelled seeds and the completeness of labelling), and the evaluation measures used for the same.

Algorithm description

The semi-supervised approach, by far, most accurately describes the real-world scenario where even though a large training data is available, only a fraction of it is labelled. The explanation pertaining to why a semi-supervised algorithm works better in general, than a supervised one, is beyond the scope of this project.
A seed set is a set $S \subseteq X$ usually with K-partitioning (i.e. every cluster has at least one representative seed point; if not, that is incomplete labelling of seed set which is also explored in this task). The initial cluster centres are calculated with this seed set and the other points are assigned corresponding clusters. Further iterations are akin to the usual K-Means algorithm.

Seeded K-Means

The seed clustering is used to initialise the K-Means algorithm. Hence, rather than initialising K-Means from K random means, the mean of the *i*th cluster is initialised with the mean of the *i*th partition S_i of the seed set. The seeds are not used in the subsequent iterations and hence there is no need to store the same (memory efficient as compared to Constrained K-Means).

Constrained K-Means

The seed clustering is used to initialise the cluster membership for the first iteration. However, for the further iterations, the membership of the original seed points is NOT recalculated and hence not changed. Computationally, it is evidently better than Seeded K-Means, but as the noise levels increase in the initial seed set, the quality of clusters decreases. This is shown with evidence in the upcoming sections.

Farthest Seeded K-Means

For the evaluation of the above algorithms with respect to incompleteness during initial seeding, another variant of Seeded K-Means called Farthest Seeded K-Means[2] is implemented as well. Basically, it employs the heuristic of iteratively choosing the remaining cluster centres (for the first iteration) as the points which are the farthest from the already obtained cluster centroids. Please note that this is NOT present in the original paper however it was still implemented because of its strong relevance.

Implementation details

Being “semi-supervised” algorithms, for the “supervised” part of the dataset (i.e. labelling), we expect the dataset to have some data points as such. However, usually that is not the case, and so a manual labelling mechanism is implemented as well. The required seeds are selected and presented to the user to label - and it is expected that those will be the true labels. If noise is to be introduced, it’d be done so systematically *after* this step of manual labelling.
Also, the 20 NewsGroups dataset (textual data) is being fetched from sklearn, and although only a handful of categories are being imported, the fetch and vectorization for the first time can take a while. Subsequent runs would be faster as it caches under *~/scikit_learn_data/20news_home* local directory.

Algorithm class files

Both the algorithms are implemented as their respective classes and following the scikit API conventions.

__init__()

The __init__() method initialises the <algorithm> object with the initial parameters like no. of clusters, seed fraction, noise fraction in the seed labelling, completeness of the seed labelling and the dataset name (just used for printing debug/info log). It also calls the _check_params() method to do some basic sanity checks of the params set - like the seed or noise fraction should be a float value between 0 & 1, and no. of clusters should be a positive number.

fit()

The fit() method takes the training set X, labels y and the initial seed set (with labels). For the first iteration, it calculates the initial centroids based off of the seed labelling, and then assigns a=cluster membership to all the other points with these centroids. For all the subsequent iterations, it recomputes cluster memberships by min L2 norm.

predict()

Based on the cluster centroids calculated during training, the points in the test set are assigned with the appropriate clusters.

visualise_results()

Calls the visualisation utils method to plot the cluster and seed details after reducing the dimensionality with PCA.

evaluate() and evaluate_fold()

Calculates the Adjusted RAND Score (ARI) and Adjusted Mutual Information Score (AMI), given the true & computed labels.

Utils

The helper functions are categorised into 4 utility scripts:

centroidutils.py

Calculates the initial and subsequent centroids & assignment of cluster membership. Also helps check if the membership has remained the same in the current iteration or not (the stopping criteria). Recomputation of the membership is controlled by a flag enable_seed_cluster_change - which is false for Constrained K-Means & true otherwise. For this control, it also takes as a parameter the indices of the original seeds.

runnerutils.py

Contains a *run_algo()* method which for a given algorithm object & dataset - separates the dataset into test & train, and for each fold, trains, tests and evaluates the performance of the algorithm.
Also contains a *run_KMeans()* method which does the above but for sklearn K-Means.

seedutils.py

The main function is to get the initial seed set. It takes the dataset and associated labels, and the tweakable parameters noise fraction, seed fraction & completeness fraction. Based on the seed fraction, it decides how many seeds it should be selecting, and randomly gets X.size*seed_fraction seeds. The selection is based on the completeness fraction - if it is 1, then there are equal no. of representative seeds from each cluster. If it is <1, then appropriate no. of classes are deliberately missed out on. However, the core K-Means algorithm still needs appropriate no. of centroids to begin with - so the remaining centroids are selected by the following criteria:
Given the partial set of centroids obtained from the *init_seed_set* with incomplete seeding, we calculate the *(n-l)* farthest points iteratively from each of these centroids - these act as the remaining cluster centres. This heuristic of selecting initial (remaining) cluster centres by maximising the distance from existing centres is proven to result in better clusters than choosing randomly. It is also a semi-supervised algorithm on its own, referred to as Farthest Seeded K-Means.[2]
If the noise fraction is 0 - then the seeds are returned as they are. If it is not - then noise_fraction*no_of_seeds are selected at random from the seed set and their labels changed to incorrect value; a manual introduction of noise.

visualisationutils.py

This is for charting the clusters as a scatterplot and evaluating the clustering quality visually. Sklearn's PCA is used to first reduce the dimensionality of the given dataset, if it has >2 features. The first plot is based off of true labels, and the second shows the computed labels; both of course alongwith the seeds selected (cross marks), the initial and final cluster centres (black squares). The plot image is also saved under the results subdirectory.

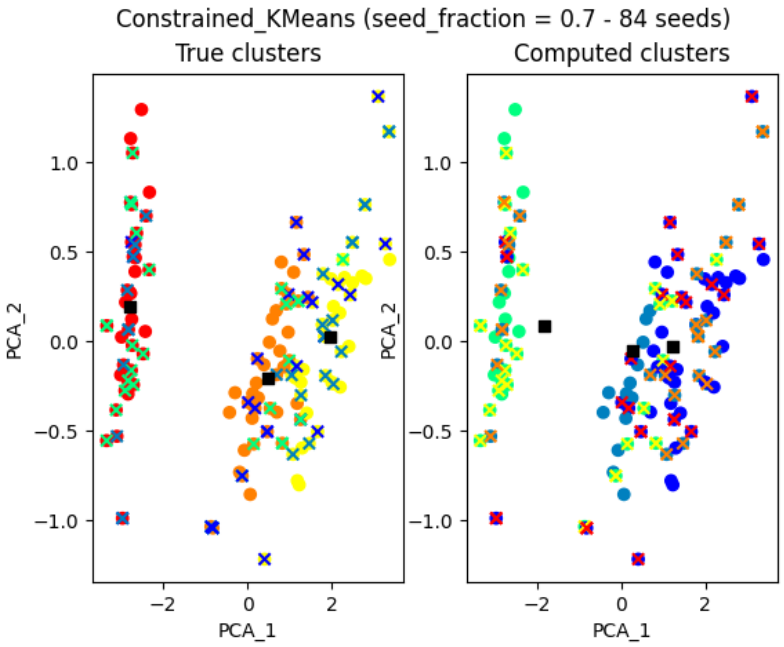


Fig. Original vs computed clusters, seeds, and cluster centres for iris dataset

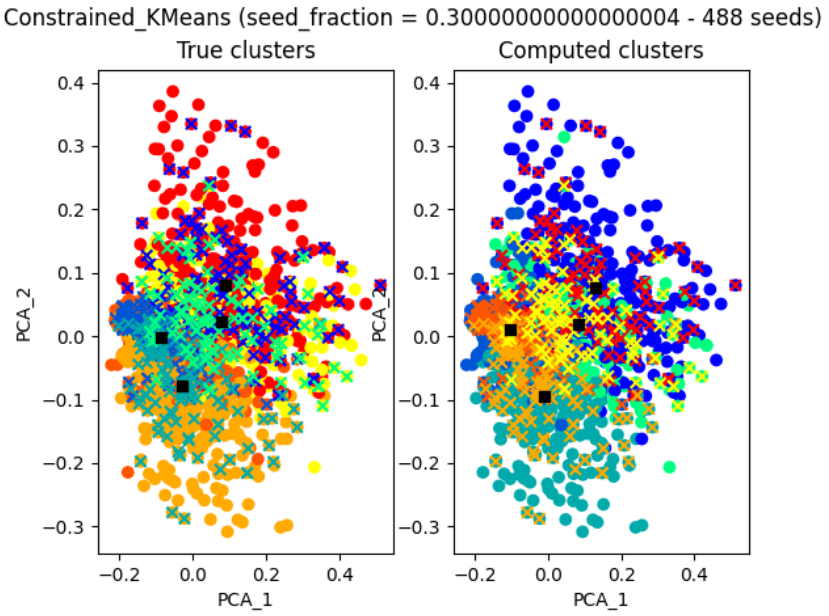


Fig. Original vs computed clusters, seeds, and cluster centres for 20 newsgroups dataset (cluster distinction not evident because of very high dimensionality of origins data - to be represented accurately with only 2 reduced dimensions).

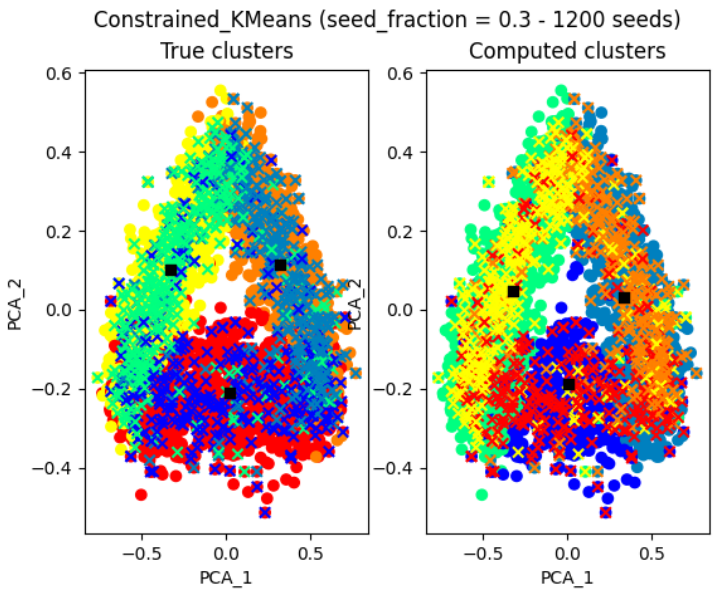


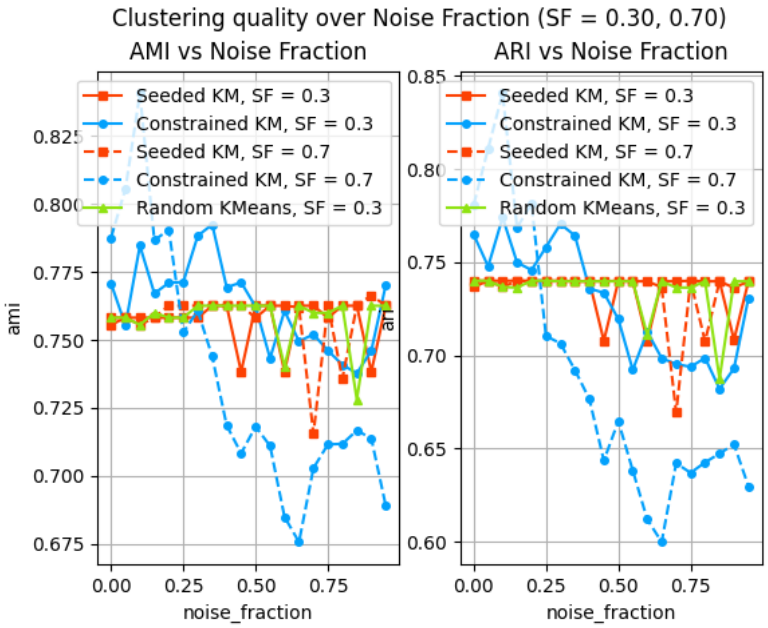
Fig. Original vs computed clusters, seeds, and cluster centres for waveform dataset

Runners

Runner files are defined for every dataset. Basically, they load the dataset & labels (if any), initialise the various algorithm objects with the necessary initial params, and return the clustering quality results (ARI & AMI).
For the *20-newsgroups* dataset, there are additional preprocessing steps to be performed as well - to transform the textual data into tabular form using tf-idf. The resultant matrix is a very sparse matrix with very high dimensions (roughly only 159 non-0 elements in a 30k+ dimensional space) - less than 0.5% non zero features. And as for the *waveform* dataset, the categorical label is encoded and the numerical values are normalised as well.

Performance Trackers

One file for each of the 3 parameters that we want to track the performance of Seeded, Constrained & other reference algorithms - noise fraction, seed fraction and completeness of labelling - we call the clustering algorithms keeping the other params constant. For a given dataset, no. of folds and no. of clusters, it calls the corresponding runner file method *cluster()*. We plot the returned AMI & ARI values for every algorithm and set of fixed params, with the varying parameter on the x-axis. The plot images are also saved under the results sub-directory.



These performance trackers are directly called from the main.py file.

Reference algorithms

The original paper used the standard randomly seeded K-Means and a previously developed semi-supervised version of K-means called COP K-means to evaluate the novel algorithms' performance. In addition to these two, we also use {} from scikit-learn for a more comprehensive comparison.
Plots are obtained for Adjusted RAND Score (ARI) and Adjusted Mutual Information Score (AMI) for every algorithm, by tweaking one hyperparameter and keeping the others constant at a time.

Datasets

The implementation was developed and tested for correctness using the classic ***Iris dataset*** imported from sklearn. The ***Waveform dataset*** from UCI repository is tried too. The real data in the dataset was normalised and the categorical label was encoded as a precprocessing step.
Additionally, for the sake of reproducing the findings of the original paper, Yahoo!'s ***20 Newsgroups dataset*** is used as well. However, this being textual data, pre-processing steps were required to make a vector-space model for each of the text documents. Please note that running all clustering algorithms, with folding and averaging with *META_COUNT* (to smoothen the ARI/AMI plots) is computationally too much if run on the full *20 Newgroups dataset*; even with an efficient implementation. Hence, only selected categories have been imported and appropriate no. of clusters specified.

Experimental results

Evaluation measures

The 2 metrics that were used to evaluate the quality of the clusters formed:

1. **Adjusted Mutual Information Score (AMI)**
   Adjusted Mutual Information (AMI) is an adjustment of the Mutual Information (MI) score to account for chance. It accounts for the fact that the MI is generally higher for two clusterings with a larger number of clusters, regardless of whether there is actually more information shared. For two clusterings and , the AMI is given as:

   AMI(U, V) = [MI(U, V) - E(MI(U, V))] / [avg(H(U), H(V)) - E(MI(U, V))]
   This metric is independent of the absolute values of the labels, so a permutation of the class or cluster label values won't change the score value in any way.

2. **Adjusted RAND Score (ARI)**
   The Rand Index computes a similarity measure between two clusterings by considering all pairs of samples and counting pairs that are assigned in the same or different clusters in the predicted and true clusterings. The raw RI score is then "adjusted for chance" into the ARI score using the following scheme:

   ARI = (RI - Expected_RI) / (max(RI) - Expected_RI)

   The adjusted Rand index is thus ensured to have a value close to 0.0 for random labelling independently of the number of clusters and samples and exactly 1.0 when the clusterings are identical.

Seed fraction

The initial intuition is that if we increase the sead fraction, i.e. have more data initially as labelled, the clustering quality would improve as it'd mean more of the "supervised" part than the "unsupervised" (assuming, of course, there is no noise in this initial labelling). Based on the results obtained, we find that it is indeed the case but only for Constrained K-Means - where we assume the initial seed labels are true & do not recompute them. For Seeded & Random K-Means, ARI/AMI is more or less constant.

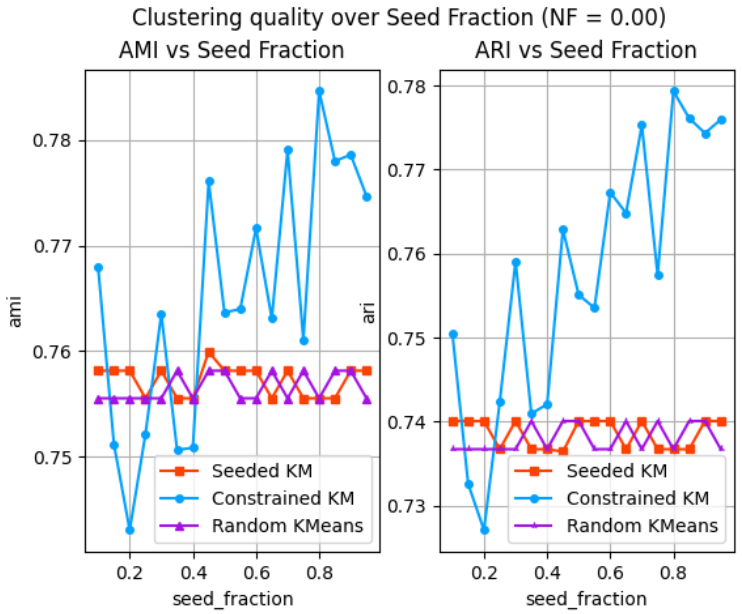


Fig. ARI/AMI vs seed_fraction for iris dataset

Noise

Seeded K-Means perform more or less consistently with increase in noise in the seed labelling. This is due to the fact that the seed cluster memberships are recalculated in subsequent iterations. Random K-Means doesn't even consider the labelled part of the seeds - the seed selection is random. So noise in seed labelling has practically no impact on this algorithm. However, since in Constrained K-Means it is NOT, so we see a quality drop (reduced ARI/AMI) with increase in badly labelled seeds.

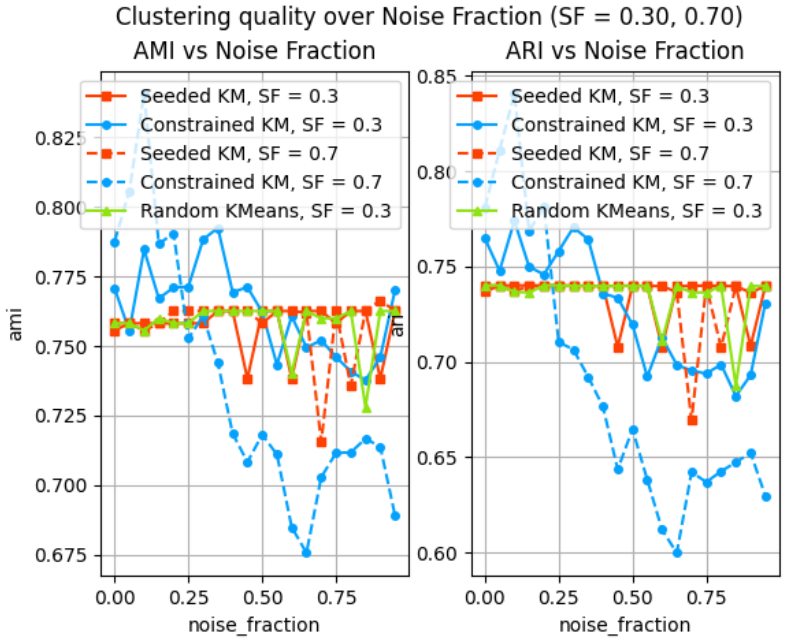


Fig. *ARI/AMI vs noise_fraction for iris dataset*

Incomplete seeding

Having incomplete seeding means to not have the seeds from every category in the initial seed set. The way this is implemented in the project is to first remove such categories from the initial dataset & then sample seeds - the reason for this is that if we select seeds randomly first & then change labels to one of the available categories, this would inadvertently introduce noise in labelling. This is something that we don't want, as then the noise_fraction we deliberately introduce will change & the results would be tainted.
As discussed in previous sections, the remaining cluster centres in the beginning are selected by choosing the points farthest from all existing cluster centres, iteratively.
K-Means does not depend on this factor as all the centroids are chosen randomly initially. In the case of Seeded K-Means, we see the clustering quality more or less stable with regards to change in *incompleteness_fraction* - this shows that the algorithm is robust enough to determine missing clusters if they're absent from the initial seed set.
Constrained K-Means also shows a similar trend, however it improves slightly with incompleteness_fraction close to 1.

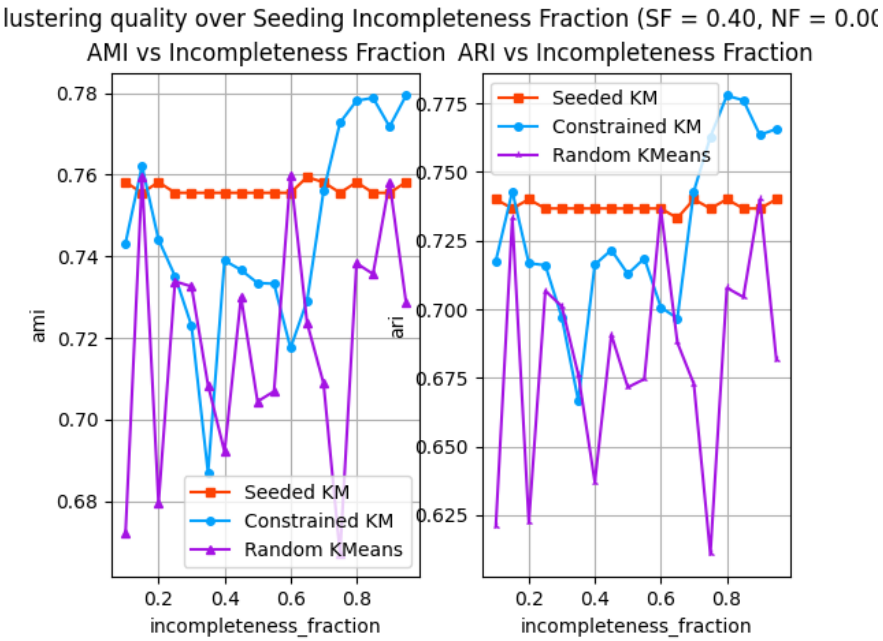


Fig. ARI/AMI vs incompleteness_fraction for iris dataset

Conclusions

From the implementation of semi-supervised algorithms (Farthest) Seeded K-Means and Constrained K-Means, we see that the quality of resultant clusters is almost always better than the unsupervised counterpart, Random K-Means. The improvement in quality is well worth the effort required to partially manually label the data, even in case where only unlabelled data is available.
Furthermore, the effect of noise and incompleteness of representation in the labelled part proves the significance of choosing the correct algorithm accordingly. When noise levels are high and there are chances of significant human-error due to manual labelling, Seeded K-Means should be chosen so that the most errors are rectified gradually over iterations. On the other end, if most categories are to be missed out in the initial labelling but whatever is

labelled will be labelled correctly (i.e. no to very low noise), then perhaps Constrained K-Means is a better option. In any case, semi-supervised methods combine the strengths of both the extreme sides of the spectrum because they dont require the pre-requisite of having fully-labelled training data as in supervised learning.

References

[1] CiteSeerX — Semi-supervised Clustering by Seeding; http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.7.9416
[2] 4487 - Semi-supervised Clustering Using Incomplete Prior Knowledge, https://link.springer.com/content/pdf/10.1007/978-3-540-72584-8_25.pdf