

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

***Frequent Mining : In-depth analysis of
Apriori Algorithms for Frequent Itemset
Mining***

GROUP PROJECT 2

**AY 2023/24 SEMESTER-1
CZ4032 - Data Analytics and Mining**

Delivered by—

Group ID: 40

Pathak Siddhant (U2023715K)

Chen Wei May (U2020687E)

Mahtolia Ronan (U2023144J)

Submitted on —

November 23, 2023

Table of Contents

Task 1: Implement Apriori Algorithm	3
Task 2: Use developed algorithm to mine frequent itemsets for 3 datasets	4
2.1 Datasets Chosen	4
2.2 Frequent-Itemset mining for Big Data - Savasere, Omiecinski, and Navathe (SON) Theorem	4
2.2.1 Implementation using Apache Spark (PySpark API)	5
2.3 Performance review and discussion	5
Task 3: Use frequent-itemsets for two clustering algorithms	6
3.1 Clustering algorithms chosen: K-Modes and Agglomerative Clustering	6
3.2 Converting dataset to frequent-itemsets to clustering features	6
3.3 Evaluation Metrics	6
3.4 Implementation for Task 3	6
3.5 Performance review and discussions	7
Task 4: Improved clustering techniques to boost performance.	8
4.1 Discussion about Task 3 algorithms	8
4.2 Improved implementation	8
5. Conclusion	10
6. Contribution Summary	10
7. References	11
8. Appendix	12
8.1 Clustering Algorithms	12
8.2 Evaluation Metrics	12
8.3 Apache Spark	12
8.4 Minimum Support Threshold Tuner results	13
8.5 Preprocessing YELP for Task 3	13
8.6 Scatter Plots	14

Task 1: Implement Apriori Algorithm

The Apriori Algorithm [7] is a popular data mining technique, used for finding frequent itemsets using the well known Apriori Principle, which is stated as follows: $\forall X, Y: (X \subseteq Y) \Rightarrow s(X) \leq s(Y)$. This is also called the anti-monotone property of support. In other words, it denotes that support of an itemset never exceeds the support of its subsets. Using a minimum support threshold, one can mine frequent-itemsets, and further expand upon them to discover strong association rules as well. For this task, we followed the given pseudocode for our implementation of the algorithm.

Procedure: Apriori Input: a transactional database $T \ll user_id_1, i_1, >, <user_id_1, i_2, > \dots >$, minimum support value, <i>minsup</i> Output: frequent-itemsets list ensure $minsup \geq 1$ $k_index \leftarrow 1$ $result_candidate_itemsets, true_frequent_k_itemsets,$ $next_k_candidate_list \leftarrow [], [], []$ $true_frequent_k_itemset_list, next_k_candidate_list \leftarrow$ $generateFrequentSingletons(T, minsup)$ while $next_k_candidate_list$ is not empty: $k_index += 1$ if $k_index = 2$: $c \leftarrow [(single,) \text{ for } single \text{ in } true_frequent_k_itemset_list]$ $result_candidate_itemsets.add(c)$ $next_k_candidate_list \leftarrow generateKTupleItemsets(c, 2)$ else: $true_frequent_k_itemset_list \leftarrow$ $countFrequentItemsets(next_k_candidate_list, T, minsup)$ $result_candidate_itemsets.add($ $true_frequent_k_itemset_list)$ $next_k_candidate_list \leftarrow$ $generateKTupleItemsets(true_frequent_k_itemset_list,$ $k_index)$ return $result_candidate_itemsets$
--

The main `apriori` implementation initialises variables for storing candidate and frequent itemsets by generating frequent singletons using `generateFrequentSingletons`. Then, it iteratively generates candidate and frequent itemsets of increasing sizes until no more candidates are found. The result is a list of lists, where each inner list contains itemsets of the same size.

`generateFrequentSingletons` generates frequent singletons (itemsets of size 1) from the input `chunk_list`. It uses a dictionary (`counter_dict`) to count the occurrences of each item in the dataset. Items that meet the minimum support threshold are added to `frequent_singletons`. It returns both the frequent singletons and the sorted list of candidate singletons.

`countFrequentItemsets` takes a list of candidate itemsets (`candidate_list`) and counts their occurrences in the dataset (`chunk_list`). It uses a defaultdict (`counter_dict`) to count occurrences efficiently. Itemsets that meet the minimum support threshold (`minsup`) are added to the `frequent_dict`. It returns a list of frequent itemsets. `generateKTupleItemsets` function generates candidate itemsets of size k from a list of frequent itemsets (`frequent_list`). It uses nested loops to combine frequent itemsets and create new candidate sets. It returns a set of candidate itemsets.

1.1 Space and Time Complexity

As visible, the space and time complexity is $O(2^d)$ where d is the number of unique items in the transactional database. The actual complexity is often considered exponential in the worst case due to the potential for an exponential number of itemsets, but it is reduced in practice by the pruning step.

Task 2: Use developed algorithm to mine frequent itemsets for 3 datasets

2.1 Datasets Chosen

Three real-world datasets of varying density and dimensionality were used to simulate different environments and evaluate time and memory efficiency of our implementation. They are as follows:

- a. **Groceries Market Basket Dataset:** This is an open source dataset obtained from Kaggle. This dataset consists of 9835 transactions by customers shopping for groceries. Memory occupied on disk is 1.7MB.
- b. **MovieLens-20M Dataset:** The MovieLens 20M dataset is a collection of 20 million ratings and 465,000 tag applications applied to 27,000 movies by 138,000 users, widely used for research in recommender systems.
- c. **Yelp Reviews Dataset:** Provided by the Yelp organisation themselves, this dataset consists of a similar schema with users providing detailed feedback about their stay alongside a numerical rating. Memory occupied on disk by raw dataset is 5.48GB.

For datasets b) and c), the transactions are created by filtering those records/items where the user-provided rating is higher than the mean/median score and aggregated later. We utilise the ratings aspect of this dataset alongside with the unique user identifiers. Different minimum support threshold values are to be used to run the simulations. For datasets with more transactions (eg: Yelp), we prefer to use a higher value compared to others. This is because the lower value thresholds are trivially satisfied and do not yield meaningful item-sets. Refer to Appendix 8.5.

2.2 Frequent-Itemset mining for Big Data - Savasere, Omiecinski, and Navathe (SON) Theorem

We cannot directly use our previously developed Apriori algorithm. The problem arises due to the large sizes of the dataset [2] [4]. Our previous algorithm hinged on the capabilities of the system to load the entire transactional database into the memory, not to mention the cost of entertaining an exponential number of possible frequent-itemsets (as discussed in the lectures) simultaneously. A naive approach to this problem is to load the data into chunks/batches and process frequent-itemsets for each chunk and then just merge all the sub frequent-itemsets to get the global frequent-itemset. A counterexample, exposing the flaws of this approach is shown below.

<u>Pass 1: Map Phase</u> Divide data into chunks For each chunk, local support threshold = $\text{global support} / \text{num_chunks} * \text{len}(\text{chunk})$ Using Apriori , find frequent itemset in that chunk Return them as $(F,)$ where F = frequent-itemsets	<u>Pass 1: Reduce Phase</u> From each mapper's $(F,)$, F is assigned to each reduce task Produces keys that appear one or more times These produced keys are candidate itemsets
<u>Pass 2: Map Phase</u> Each map task takes input from phase 1 reduce task and selects a chunk of the database. Every map task will have all the candidate itemsets. It counts the occurrence of each candidate itemset among the baskets in that selected chunk of the dataset. Output of the map task is (C, v) where C : candidate itemset and v : support among the baskets input to this mapper	<u>Pass 2: Reducer Phase</u> Each reduce task takes itemset from the above mappers. Calculates sum of associated values (support) for each itemset. If $\text{support}(\text{itemset}) \geq \text{global support}$: Add to final global frequent itemsets Else: Continue till end of results

To solve rule and itemset mining problem using data partitioning, we came across something known as the **SON algorithm [3]**. The SON algorithm is an adaptation of the Apriori algorithm to work efficiently on distributed systems, such as those that use MapReduce. It is a distributed data mining algorithm designed to find frequent itemsets in large datasets.

The SON algorithm can be thought of as 2 passes of MapReduce. MapReduce has two passes: the first pass is to find candidate itemsets and the second later pass is to find correct

itemsets. The pseudocode of the algorithm is provided above.

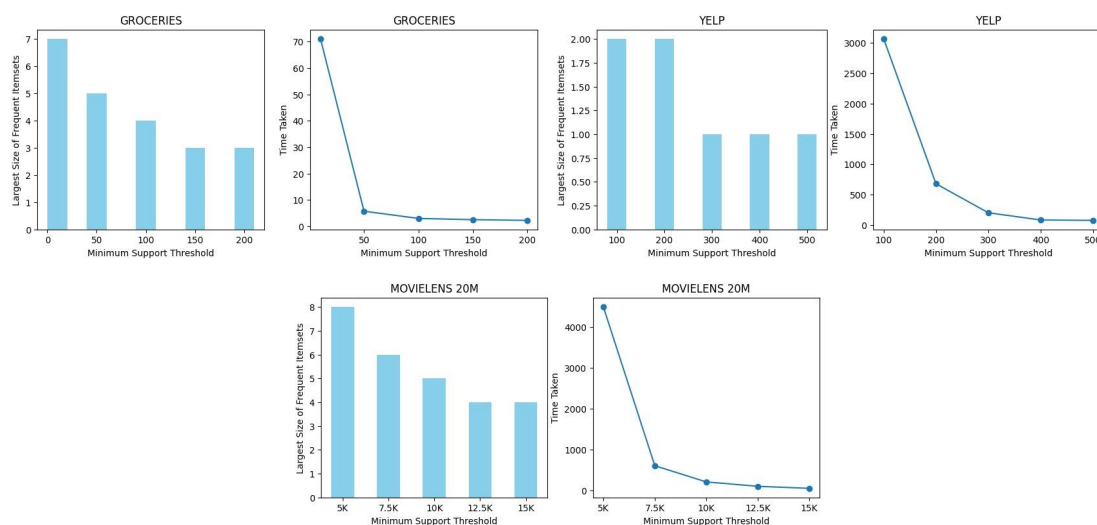
2.2.1 Implementation using Apache Spark (PySpark API)

To exploit the power of current modern multi-core multi-threaded CPUs, we use Apache Spark framework [5] to implement the aforementioned algorithm. We used Pyspark API library for the scope of this task. Spark helps in faster partition and implements the map-reduce logic user the master-slave architecture instead of using traditional Pandas library which has limited capability for scalable solutions [6]. To monitor the performance and scheduling of different jobs, we utilise Spark UI.

2.3 Performance review and discussion

Dataset Name	Minimum Support Threshold	Number of Itemsets	Largest Size of Frequent Itemsets	Time taken (in seconds)
Groceries (38765)	10	20K+	7	71.12
	50	1950	5	5.73
	100	552	4	2.99
	150	270	3	2.51
	200	158	3	2.26
MovieLens-20M (9995410)	5K	20K+	8	4494.31
	7.5K	8151	6	602.55
	10K	2019	5	206.28
	12.5K	724	4	98.74
	15K	26	4	48.97
Yelp (430678)	100	977	2	3070.49
	200	446	2	677.89
	300	244	1	200.01
	400	153	1	81.26
	500	99	1	75.05

Graphs below show analysis of the time taken as well as the largest size of frequent itemsets and their variations plotted against different support thresholds as a line and bar plot respectively.



The graphs suggest that for both the Groceries and Movielens 20M datasets, as the minimum support threshold increases, the size of the largest frequent itemsets decreases, which indicates that fewer itemsets meet the criteria of being 'frequent' as the threshold becomes more stringent. Additionally, there is a clear inverse relationship between the minimum support threshold and the time taken for computation across all datasets; higher thresholds result in quicker computations. This could be due to the reduced number of itemsets to evaluate at higher thresholds. For the Yelp dataset, the drop in time taken is particularly steep as the threshold increases from 100 to 200, which may indicate that fewer complex itemsets are being considered beyond a certain support level.

Task 3: Use frequent-itemsets for two clustering algorithms

For this task, the Breast Cancer Wisconsin dataset and the NASA KC2 [10] dataset were used. The Breast Cancer Wisconsin (Diagnostic) dataset is a widely used medical dataset that includes measurements from digitised images of fine needle aspirate (FNA) of breast masses, helping in the diagnosis of breast cancer. The KC2 dataset is a software engineering dataset used for defect prediction, containing metrics and defect data from a NASA project, aiding the software quality assessment and improvement

3.1 Clustering algorithms chosen: K-Modes and Agglomerative Clustering

3.1.1 KModes

KModes [9], an adaptation of KMeans, is used to classify categorical data, (KMeans works for numerical data.) KModes works by utilising Hamming distance to measure dissimilarities between data points based on cluster attributes. Clusters are defined based on matching categories between data points.

3.1.2 Agglomerative Clustering

Agglomerative Clustering is a type of hierarchical clustering. It starts with individual data points and gradually merges them into successively larger clusters. The theory is to merge until all similar clusters become one or a certain number of clusters is met. It finds the distance between clusters using a user-given metric (default is Euclidean).

3.2 Converting dataset to frequent-itemsets to clustering features

To prepare the data, we first find the frequent itemsets using the apriori algorithm imported from the popular mlxtend python library. For the scope of this project, we used instances where the number of frequent itemsets is more than 1, since that allows visualisation in a two-dimensional space (dimensionality reduction techniques like PCA are applied if needed).

After obtaining frequent itemsets from the dataset, we convert them to a new dataframe which is a binary dataframe where the values are decided for each row whether each subset of the frequent itemset is True or not. This sparse binarised dataset is then used for clustering as the input.

3.3 Evaluation Metrics

We used 3 main evaluation metrics, namely: Accuracy, Silhouette Coefficient and Davies-Bouldin Score [8]. Refer to Appendix 8.2 for more details.

3.4 Implementation for Task 3

The figure shows the pseudocode of our implementation of a method named `task3` where the inputs are the prepared dataset, along with the ground truth labels and clustering algorithm of choice. The

utility functions serve the purpose as named: `do_clustering` performs clustering based on the choice of algorithm, `remap_labels` points out right permutation of the predicted cluster annotation with the ground truth label for easier understanding, `evaluate_clusters` and `plot_clusters` are used for metric-based evaluation as well as visualising the clusters in 2D and 3D scatterplots (PCA used to reduce dimensionality of the dataset)

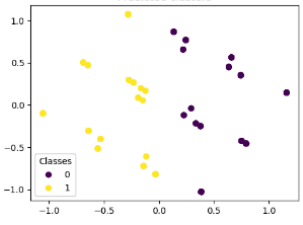
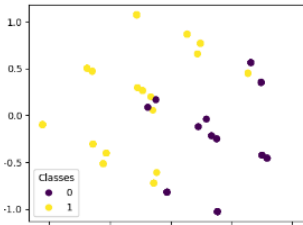
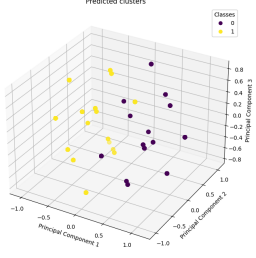
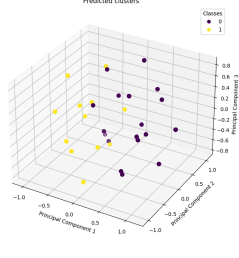
Procedure: Task 3
Input: binary database <i>df</i> <<uid, T, F, F, T...>, ground label <i>target</i> , algorithm name <i>algo</i>
Output: accuracy, plots of clusters
<pre> n ← unique(target) clusters ← do_clustering(df, algo, n_clusters = n) accuracy, remapped_labels ← remap_labels(clusters, target) evaluate_clusters(remapped_labels, target, df) plot_clusters(df, remapped_labels, target)s </pre>

3.5 Performance review and discussions

Dataset Name	KModes			Agglomerative Clustering		
	Accuracy	Silhouette Score	Davies Bouldin Score	Accuracy	Silhouette Score	Davies Bouldin Score
Breast Cancer	0.7540	0.3057	1.5103	0.6784	0.3096	1.5727
NASA KC2	0.7375	0.6921	0.5320	0.7471	0.6959	0.4895

From the table, it is evident that K-Modes clustering achieved a higher accuracy for the Breast Cancer dataset than Agglomerative Clustering, suggesting it may be better suited for this type of data. Conversely, Agglomerative Clustering outperformed K-Modes in terms of the Silhouette Score for the NASA KC2 dataset, indicating it was able to find more cohesive and separated clusters for this particular dataset. Refer to Appendix 8.6 for detailed comparison of predicted and actual clustering.

Dataset: Breast Cancer

	KModes	Agglomerative Clustering
2D Scatterplot		
3D Scatterplot		

Dataset: NASA KC2

	KModes	Agglomerative Clustering
2D Scatterplot		
3D Scatterplot		

Task 4: Improved clustering techniques to boost performance.

The aim of this task is to view and analyse the performance of clustering algorithms in Task 3 and based upon them as baselines, suggest improvements in the pseudocode.

4.1 Discussion about Task 3 algorithms

As seen in Task 3, the algorithms used do not yield significant results. After visually inspecting their allocation of clusters, we saw that the algorithms fail to cluster data points in a complementary fashion and yet identify the major same ones.

4.2 Improved implementation

Our improved implementation consists of two major changes in the logic from that used in Task 3. We implemented a **Parameter Tuner** and a **VotingClassifier**. More details explained in the next sections.

4.2.1 Motivation

Different minimum support thresholds in frequent itemset mining produce a variety of outcomes: lower thresholds often result in a greater number of frequent but perhaps less significant itemsets, whereas higher thresholds yield fewer itemsets that are generally more relevant, as they represent items more commonly found together. The resulting frequent itemsets then become the features for clustering, where the number of features can significantly affect the clustering algorithm's performance. More features can create complex and potentially insightful clusters but raise the risks of overfitting and computational demands; fewer features may lead to oversimplification and less nuanced clusters.

In light of this, both the KModes and Agglomerative Clustering algorithms demonstrate a capacity to interpret the data independently, suggesting that an ensemble method could be advantageous. A voting classifier ensemble, which aggregates predictions from multiple models, aims to enhance accuracy and stability, capitalising on the distinct strengths of each algorithm. It typically results in superior performance compared to individual models, as it mitigates the risk of overfitting and offers more reliable predictions. In such ensembles, any ties in the voting process are resolved at random.

4.2.2 Pseudocode

The figure below provides the pseudocode of the parameter tuner we implemented to loop through all the possible values for all the chosen list of algorithms. This leads to an exhaustive grid search algorithm, and chooses the best value of minimum support threshold where all the models are performing, since the voting classifier needs input from all the individual models. The search space of the threshold is limited to (0.2, 0.6] with a step of 0.01 due to computational and time constraints

Procedure: minsup_tuner
Input: database $df \ll uid, T, F, T, \dots \gg$, ground labels $target$, algorithm list $algo$, minsup search list m_list
Output: best_minsup_value
<pre> n, cutoff ← unique(target), 1 for minsup in m_list: if minsup > cutoff: break frequent_itemsets = get_frequent_itemsets(df, minsup) if number of frequent_itemsets is 0: cutoff = minsup; break for algo in algorithm: best_acc, best_minsup ← 0, 0 clusters ← do_clustering(df, algo, n_clusters = n) accuracy ← remap_labels(clusters, target) if accuracy > best_acc: best_acc = acc; best_minsup = minsup best_minsup_for_all ← highest accuracy for ALL algorithms plot_all_minsup_accuracies(best_minsup, best_acc, best_minsup_for_all) return best_minsup_for_all </pre>

```

○○○

class VotingClassifier:
    def voting(self, individual_predictions, num_classes, num_samples, target):
        votes = self.hard_voting(individual_predictions, num_samples)
        acc, votes = remap_labels(votes, target)
        return votes

    def hard_voting(self, individual_predictions, num_samples):
        new_votes = []
        for i in range(num_samples):
            cur_votes = self.get_cur_votes(individual_predictions, i)
            most_voted = np.argmax(np.bincount(cur_votes))
            new_votes.append(most_voted)
        return new_votes

    def get_cur_votes(self, individual_predictions, idx):
        cur_val = []
        for estimator_class in self.estimator_names:
            cur_val.append(individual_predictions[estimator_class][idx])
        return cur_val

```

We implemented a voting classifier **from scratch**. We cannot use the standard scikit-learn implementation since that focuses solely on estimators, and clustering algorithms do not classify as estimators [8]. The code snippet below showcases the outline for the `VotingClassifier` class written by us, to embody an ensemble learning solution to combine outputs from each clustering algorithm and use hard-voting (i.e. all algorithms are weighted equally, weights = 1).

For the list of clustering algorithms used, we preferred to use more than two algorithms to avoid tie-based situations (if they arise, solved randomly).

For the scope of this task, other than two algorithms we looked in Task 3, we incorporate two more as part of novelty solution:

- Gaussian Mixture Model:** It is a probabilistic clustering method that assumes data points are generated from a mixture of several Gaussian distributions with unknown parameters, enabling the capture of cluster shapes that are more complex than those identified by k-means.
- Spectral Clustering:** It is an algorithm that uses eigenvalues of a similarity matrix to reduce dimensionality before applying a clustering method like k-means to group data points into clusters based on their similarity.

Support for other types of clustering algorithms such as Ward Linkage is also provided but not examined under this section.

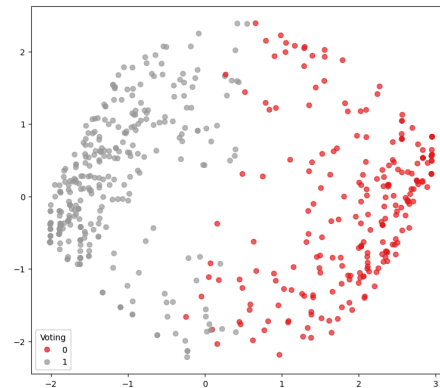
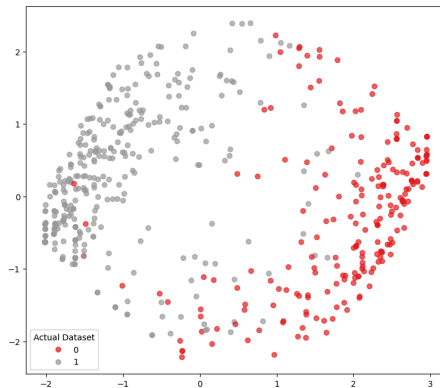
4.2.3 Performance Analysis

SS refers to the Silhouette Score and DB refers to the Davies Bouldin Score in the table below [8]. The highest performances are highlighted in red and the second highest are underlined. To compare the performance of our new approach, we have also provided the benchmarks produced from the previous task in the table below.

In the results of task 4, it can be noted that despite the accuracy being above 90%, the Silhouette Coefficient is quite low. This is due to the fact that the clusters have many overlapping data points and

as mentioned above, a Silhouette Coefficient of 0 implies overlapping clusters which is consistent with the score that we got (refer to Figure below for visualisation of the clusters).

Dataset Name	Before (Task 3)						After (Task 4)		
	KModes			Agglomerative			(KModes + Agglomerative + Gaussian + Spectral)		
	Acc.	SS	DB	Acc.	SS	DB	Acc.	SS	DB
Breast Cancer	<u>0.7540</u>	0.3057	1.5103	0.6784	<u>0.3096</u>	<u>1.5727</u>	0.9262	0.3806	1.3783
NASA KC2	0.7375	0.6921	<u>0.5320</u>	<u>0.7471</u>	<u>0.6959</u>	0.4895	0.8142	0.7690	0.3402



5. Conclusion

The Apriori Algorithm is useful when trying to mine frequent itemsets and hence can be used in commercial settings to recommend items to customers based on the items they purchase.

We then ran clustering on a couple of datasets where we used their frequent itemsets as features, we converted the data to binary form. We ran into a problem where the clustering labels were mismatched with the ground truth. To solve that we implemented a label matcher function. The results weren't very accurate and so we developed an algorithm to improve the performance. This was the voting classifier of task 4 and it significantly boosted the accuracy of the clustering. It works similarly to ensemble where it takes into account the predictions of multiple clustering algorithms.

6. Contribution Summary

The following is the contribution of the members of our group which led to the success of this project:

1. Pathak Siddhant: Apriori for Big Data, SON Algorithm, Novelty Improved Clustering (Tasks 2, 4)
2. Mahtolia Ronan: Clustering for Itemsets, Data Analysis (Task 3)
3. Chen Wei May: Apriori Algorithm, KModes ,Evaluation Metrics and Visualisations (Tasks 1,3)

7. References

- [1] V. Chauhan and M. Sharma, "A REVIEW: MAPREDUCE AND SPARK FOR BIG DATA ANALYTICS. 2016.
- [2] A. K. Luhach, D. S. Jat, K. H. Bin Ghazali, X.-Z. Gao, and P. Lingras, Eds., *Advanced Informatics for Computing Research: 4th International Conference, ICAICR 2020, Gurugram, India, December 26–27, 2020, Revised Selected Papers, Part I*, vol. 1393. in Communications in Computer and Information Science, vol. 1393. Singapore: Springer Singapore, 2021. doi: [10.1007/978-981-16-3660-8](https://doi.org/10.1007/978-981-16-3660-8).
- [3] A. Savasere and E. Omiecinski, "An Efficient Algorithm for Mining Association Rules in Large Databases".
- [4] K. Sampath Kini, Department of Computer Science and Engineering, NMAMIT, Nitte, Karkala - 574110, Karnataka, India;, K. Karthik Pai, and Department of Information Science and Engineering, NMAMIT, Nitte, Karkala - 574110, Karnataka, India;, "Determining Frequent Item Sets using Partitioning Technique for Large Transaction Database," *Indian Journal of Science and Technology*, vol. 12, no. 3, pp. 1–4, Jan. 2019, doi: [10.17485/ijst/2019/v12i3/140766](https://doi.org/10.17485/ijst/2019/v12i3/140766).
- [5] R. Agrawal, R. Srikant, H. Road, and S. Jose, "Fast Algorithms for Mining Association Rules".
- [6] T. Drabas, D. Lee, and H. Karau, *Learning PySpark: build data-intensive applications locally and deploy at scale using the combined powers of Python and Spark 2.0*. Birmingham Mumbai: Packt, 2017
- [7] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, Washington D.C. USA: ACM, Jun. 1993, pp. 207–216. doi: [10.1145/170035.170072](https://doi.org/10.1145/170035.170072).
- [8] Pedregosa et al, "API Reference — scikit-learn 0.24.2 documentation," *scikit-learn.org*, 2011. <https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics>
- [9] N. de Vos, "nicodv/kmodes," *GitHub*, Dec. 05, 2021. <https://github.com/nicodv/kmodes>
- [10] J. Sayyad Shirabad and T. J. Menzies, "OpenML," *www.openml.org*, 2005. <https://www.openml.org/search?type=data&sort=runs&id=1056&status=active> (accessed Nov. 23, 2023).

8. Appendix

8.1 Clustering Algorithms

8.1.1 *KModes*

The Hamming distance formula is shown in the figure. By this logic, it can be seen that each cluster is categorised by a feature which is the most frequent in that cluster, hence called mode.

K-Modes reiterates, calculating dissimilarities between data points and recalculating the modes in each cluster, until all the clusters are stable and there is little to no change in them. So the goal is to minimise the dissimilarities between data points in a cluster.

$$D_H = \sum_{i=1}^k |x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

8.1.2 *Agglomerative Clustering*

It decides closes based on 3 different types of linkages: Single, Complete, Average. Single Linkage is the distance between the closest points in the clusters. Complete is the distance between the two furthest points in each cluster. Average as the name suggests, is the average distance between the points in the clusters.

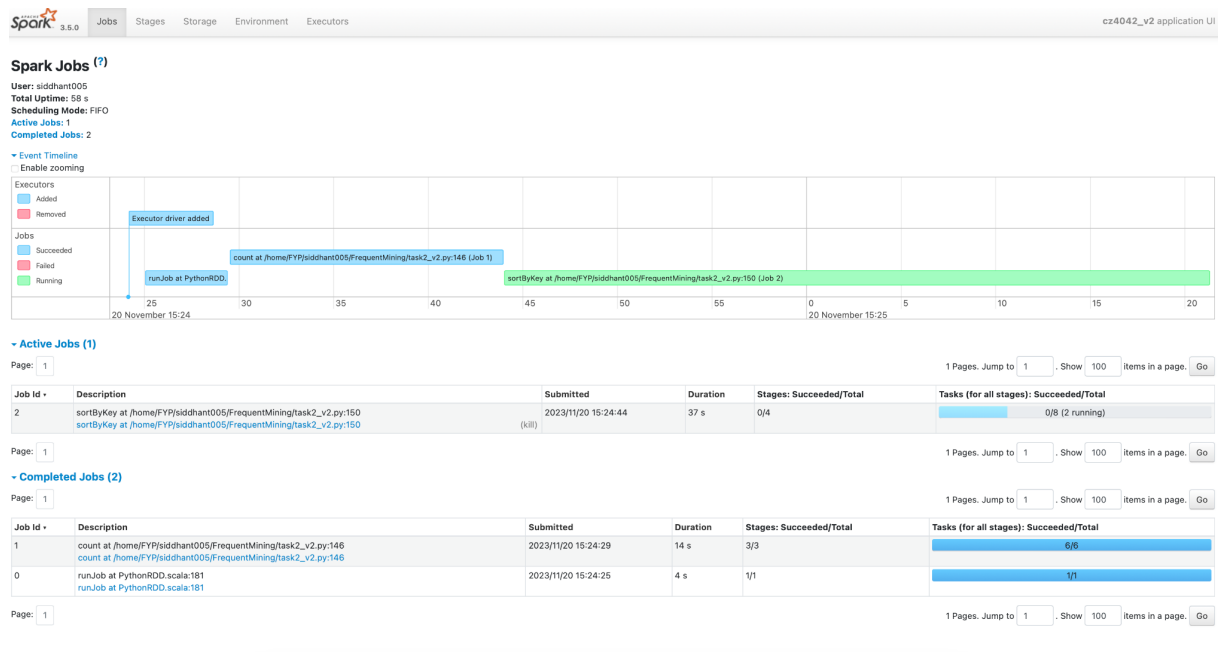
8.2 Evaluation Metrics

To compute the accuracy we used a remap function that checked all permutations of the cluster labels and returned the one that matched the actual labels.

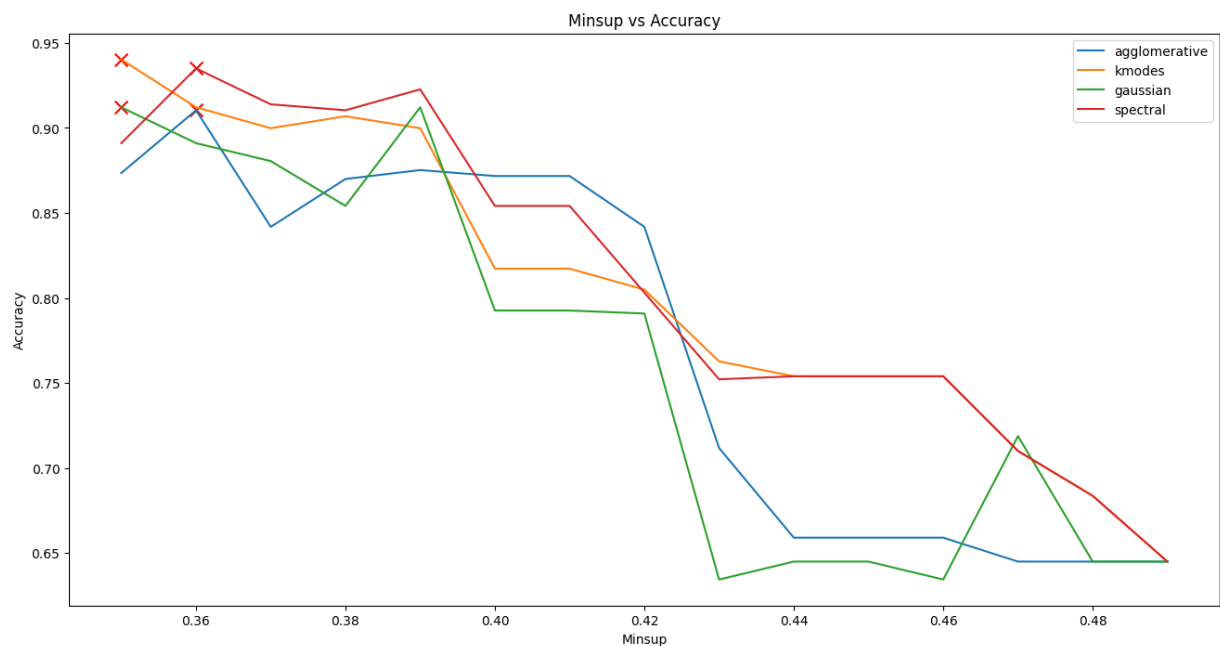
1. **Accuracy** was calculated by dividing the number of correct predictions by the total number of predictions.
2. **Silhouette Score** is a measure of how well a data point belongs to its own cluster and is separate from other clusters. It ranges from -1 to 1, where 1 implies dense and separate clusters and 0 implies overlapping clusters.
3. **Davies-Bouldin Score** is a ratio of the within-cluster distance to the between-cluster distance. It has no predefined range, but the minimum is 0. A lower score indicates compact and well-defined clusters.

8.3 Apache Spark

The Spark UI, an integral feature of Apache Spark, offers an extensive view of the Spark ecosystem. It covers details about nodes and executors, settings and properties of the environment, status of current and past jobs, including those that have finished, stopped, or failed, as well as query plans and additional insights. It runs directly on your localhost (default port 4041) when you submit a job/series of jobs.



8.4 Minimum Support Threshold Tuner results



8.5 Preprocessing YELP for Task 3

We use PySpark API to load and preprocess dataset from the 5GB reviews.json file to avoid OutOfMemory Error. Then we use the above code to map the user IDS to the business IDs.

```

from pyspark import SparkContext
import json
import csv
import sys

business_file_path = "/data/yelp_dataset/business.json"
review_file_path = "/data/yelp_dataset/review.json"
output_file_path = "/data/yelp_dataset/user_business.csv"
state = "NV"

sc = SparkContext.getOrCreate()
business_list = sc.textFile(business_file_path).map(lambda row: json.loads(row))\
    .map(lambda json_data: (json_data["business_id"], json_data["state"]))\
    .filter(lambda kv: kv[1] == state).map(lambda line: line[0]).collect()

user_list = sc.textFile(review_file_path).map(lambda row: json.loads(row))\
    .map(lambda json_data: (json_data["user_id"], json_data["business_id"]))\
    .filter(lambda kv: kv[1] in business_list).collect()

with open(output_file_path, mode='w', newline='') as result_file:
    # write output to csv file
    result_writer = csv.writer(result_file)
    result_writer.writerow(["user_id", "business_id"])
    result_writer.writerows([i for i in user_list])
    result_file.close()

```

Python

23/11/22 22:24:33 WARN Utils: Your hostname, Siddhant-MacBook-Air.local resolves to a loopback address: 127.0.0.1; using 192.168.0.156 instead (on interface en0)

23/11/22 22:24:33 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address

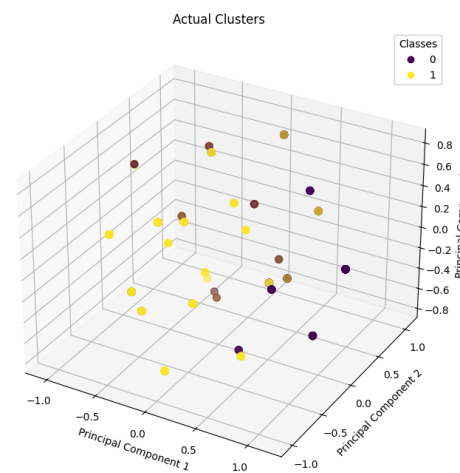
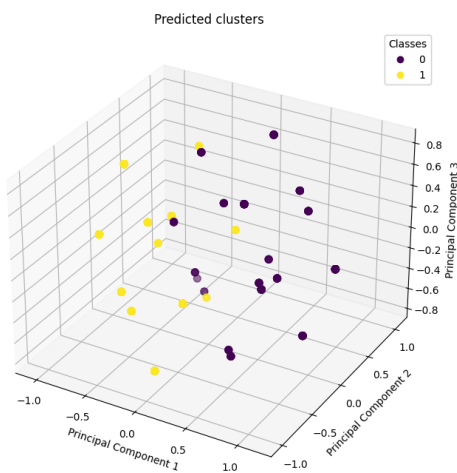
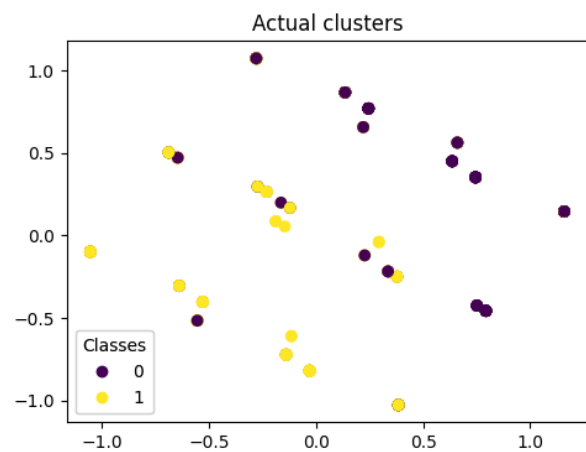
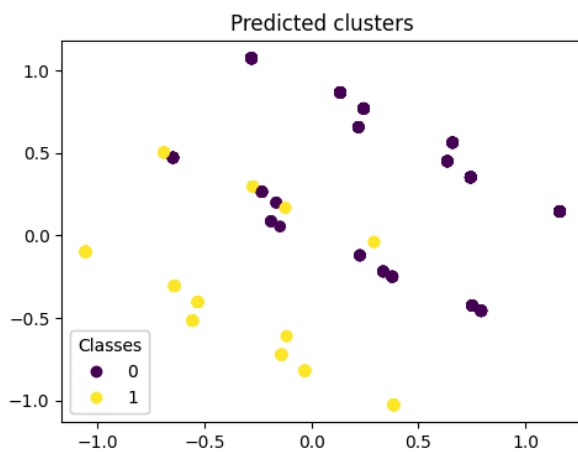
Setting default log level to "WARN".

To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).

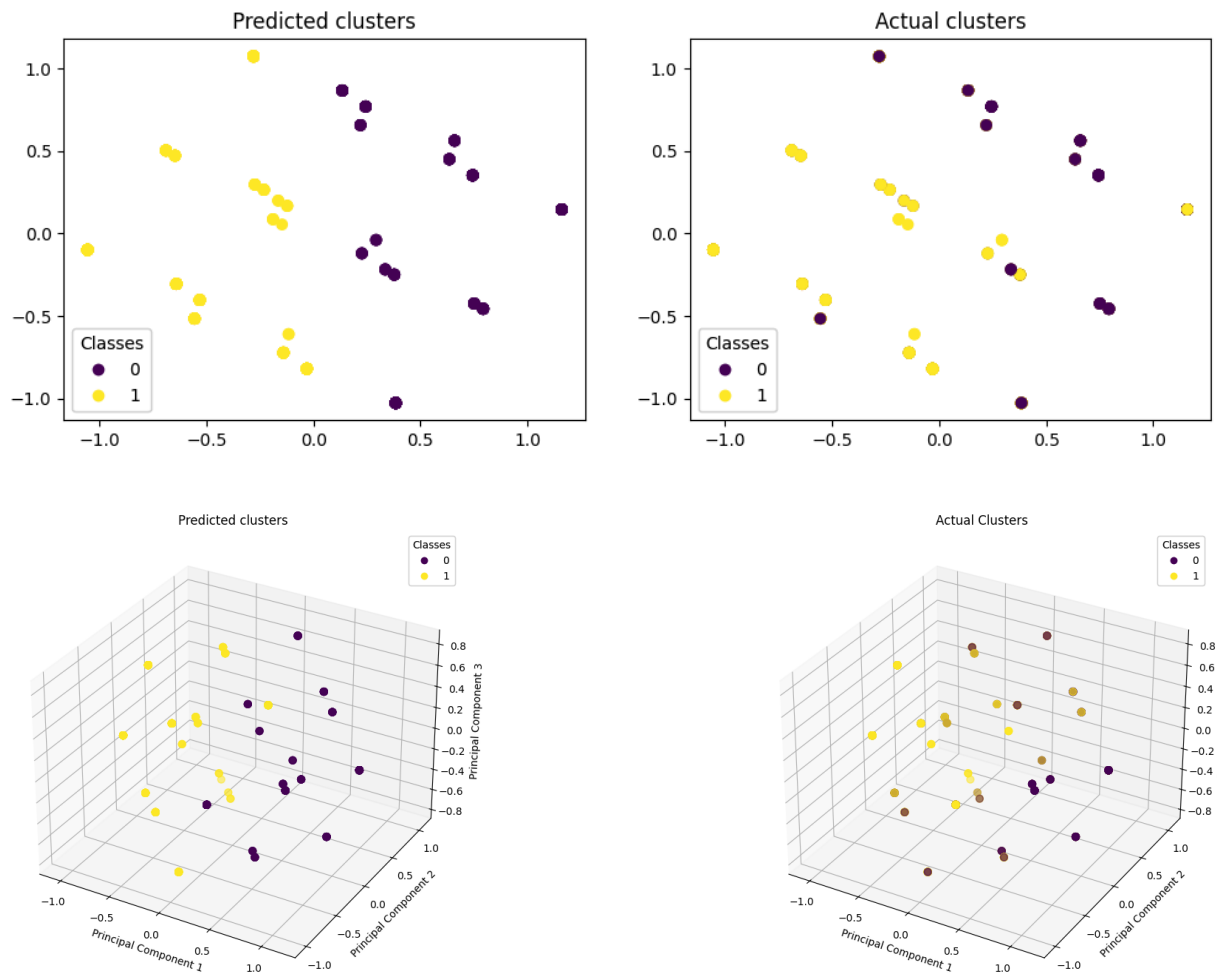
23/11/22 22:24:33 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

8.6 Scatter Plots

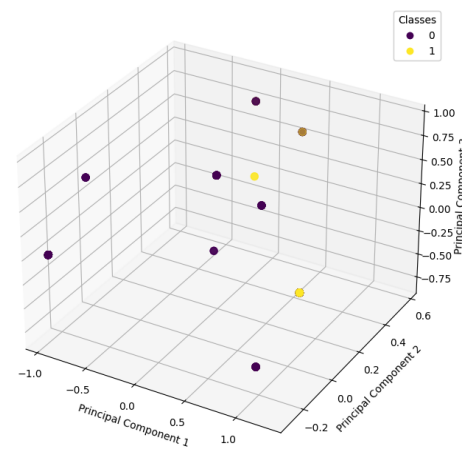
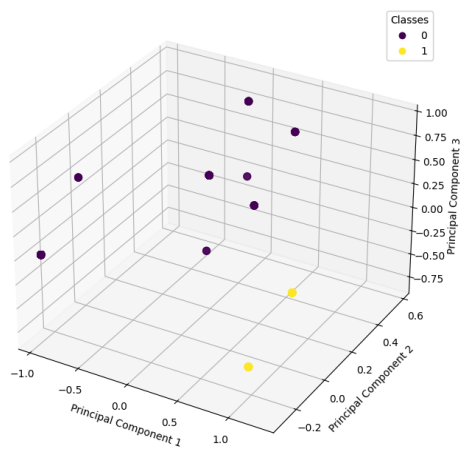
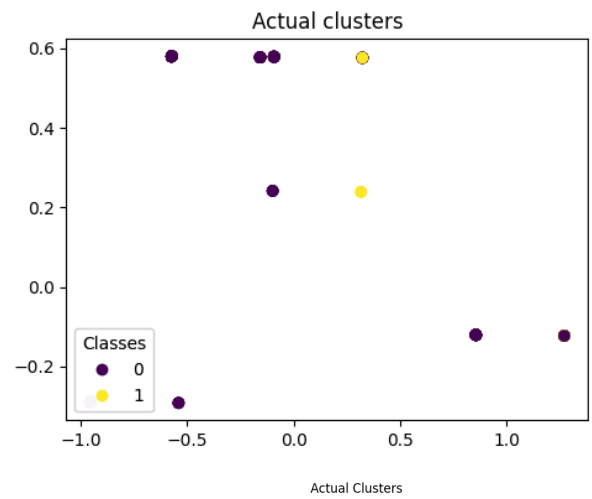
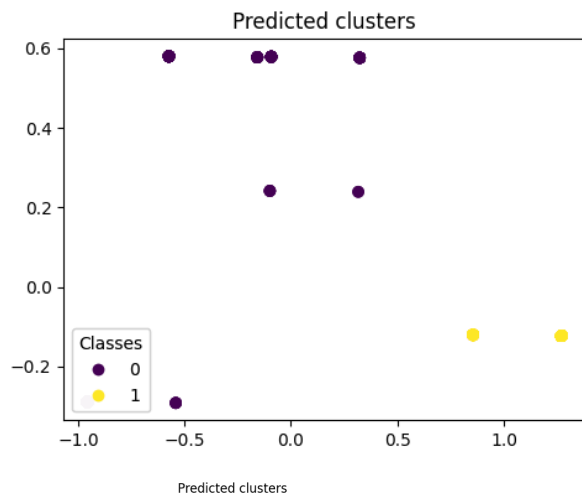
Task 3: Breast Cancer Wisconsin dataset (Agglomerative - 2D, 3D)



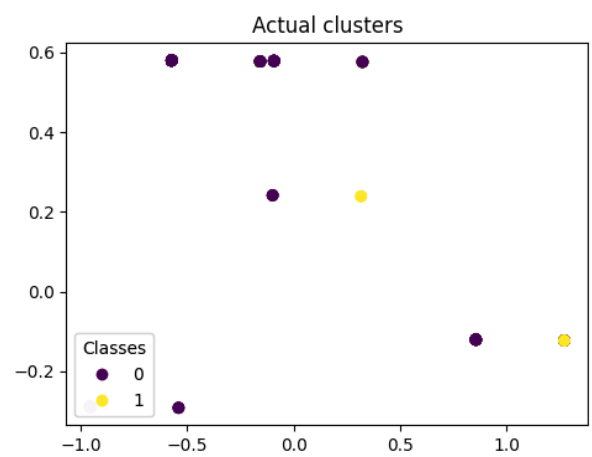
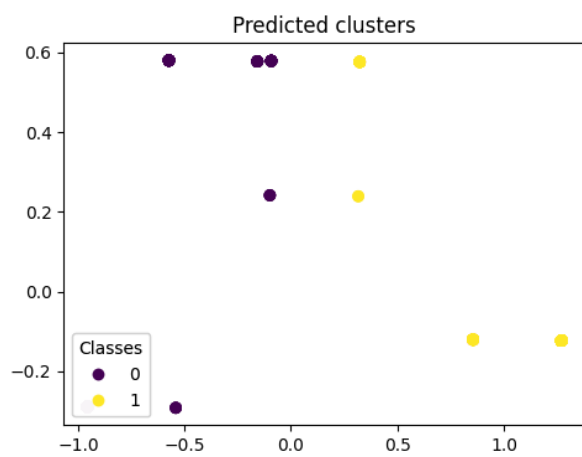
Task 3: Breast Cancer Wisconsin dataset (KModes - 2D, 3D)

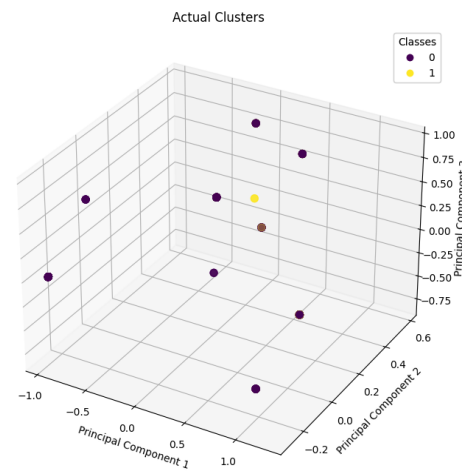
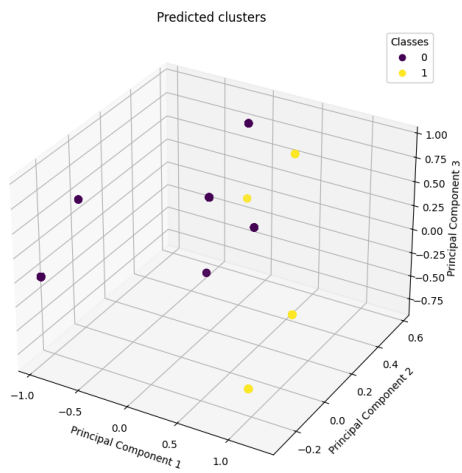


Task 3: NASA KC2 dataset (Agglomerative - 2D, 3D)

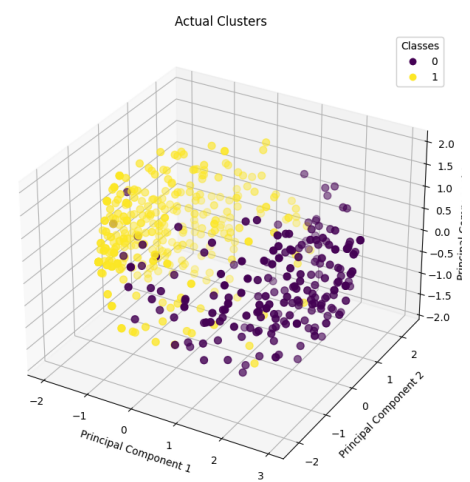
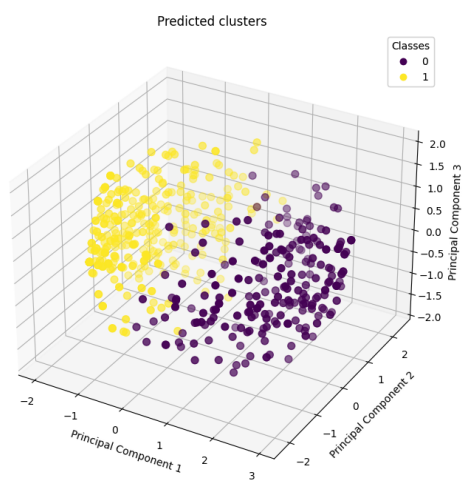
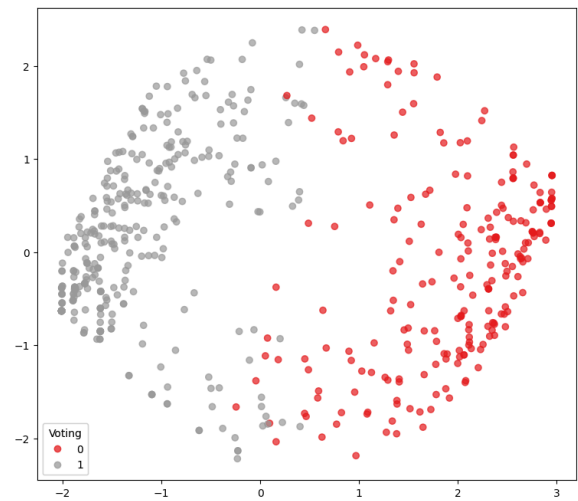
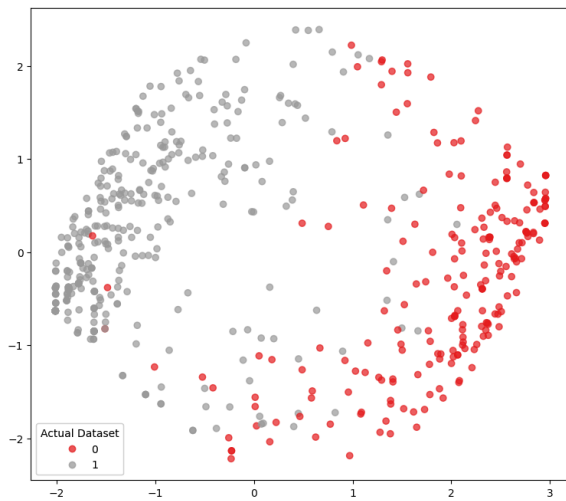


Task 3: NASA KC2 dataset (KModes - 2D, 3D)





Task 4: Breast Cancer Wisconsin Dataset (2D, 3D)



Task 4: NASA KC2 Dataset (2D, 3D)

