**New Dolphin Swarm Clustering Algorithm**

Xintong Zhu

# ABSTRACT

This project implements a new bio-inspiring clustering algorithm based on dolphin swarm intelligence. The algorithm models the clustering process by simulating dolphins’ special behaviors including echolocation, labor assignment, and cooperation while predation. The algorithm considers cluster centroids as dolphins, and data points, which need to be assigned, as prey In order to model and simulate predation, the algorithm divides predation into four stages: individual search, call, reception, and predate. Dolphins, i.e., cluster centroids, move towards preys, i.e., assigned datapoints, based on a certain mathematical function. 11 datasets are tested, applying both cosine similarity and Euclidean distance. Confusion matrices, recall, precision, and accuracy are calculated and provided, and clustering results are visualized and displayed. Based on the evaluation, I deduce that this clustering algorithm is good at clustering linear-shape datasets and is also good at processing and predicting labels for datasets that do not have clear cluster pattern at the very first beginning. One disadvantage so far discovered is that the visualization is not very good to see for small size dataset. **Keywords:** Bio-inspiring, Clustering, Dolphin, Swarm Intelligence, Echolocation

# INTRODUCTION

Swarm intelligence has been a popular topic in artificial intelligence and machine learning fields since its appearance in the 1980s. Many algorithms have been designed based on swarm intelligence principles which are broadly applied in fields such as data analytics and deep learning. Among these, biology-inspired algorithm is an essential division. One famous example can be the Bird Flocking Algorithm which has been further studied and improved since its first appearance. Indeed, in nowadays worlds with big data generated per million second, more efficient and accurate data analytics algorithms are needed, and swarm intelligence is undoubtedly one of the most reliable models to study on.

This paper studies dolphins, one of the smartest animals in the world. Many people have probably once seen the scene that dolphins jumped out of the sea level one by one. Similar to birds, dolphins are also social animals. With IQ higher than 100, they know how to cooperate with each other and assign labor while predating, utilizing echolocation. This paper focuses on this point and designs a new biology-inspiring clustering algorithm. The clustering model is based on simulation dolphins’ swarm intelligence while predating and echolocation by first studying the dolphin swarm algorithm in previous papers [1].

In Section 3, training datasets and testing datasets are described, and the preprocessment of datasets are discussed. In Section 4, the methodology is explained in detail, including the model and the algorithm. In Section 5, the experiments and evaluation results are included. And in Section 6, conclusion and future work are discussed.

# DATASETS

This section introduces datasets this project used in testing and evaluation phase. Two types of training datasets are used: pure-number datasets and pure-text datasets. This section then explains how each of the two types of training datasets are preprocessed for further result display and performance evaluation.

## **Dataset Description**

*Iris Data Set:* This is perhaps the best known database to be found in the pattern recognition literature. Fisher’s paper is a classic in the field and is referenced frequently to this day. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant.

Attribute includes: 1) sepal length in *cm* 2) sepal width in *cm* 3) petal length in *cm* 4) petal width in *cm* 5) class (Iris Setosa / Iris Versicolour / Iris Virginica)

*HTRU2 Data Set:* HTRU2 is a data set which describes a sample of pulsar candidates collected during the High Time Resolution Universe Survey. Pulsars are a rare type of Neutron star that produce radio emission detectable here on Earth. They are of considerable scientific interest as probes of space-time, the inter-stellar medium, and states of matter. As pulsars rotate, their emission beam sweeps across the sky, and when this crosses our line of sight, produces a detectable pattern of broadband radio emission. As pulsars rotate rapidly, this pattern repeats periodically. Thus pulsar search involves looking for periodic radio signals with large radio telescopes. Each pulsar produces a slightly different emission pattern, which varies slightly with each rotation. Thus a potential signal detection known as a ’candidate’, is averaged over many rotations of the pulsar, as determined by the length of an observation. In the absence of additional info, each candidate could potentially describe a real pulsar. However in practice almost all detections are caused by radio frequency interference (RFI) and noise, making legitimate signals hard to find.

Machine learning tools are now being used to automatically label pulsar candidates to facilitate rapid analysis. Classification systems in particular are being widely adopted, which treat the candidate data sets as binary classification problems. Here the legitimate pulsar examples are a minority positive class, and spurious examples the majority negative class. At present multi-class labels are unavailable, given the costs associated with data annotation. The dataset contains 16,259 spurious examples caused by RFI/noise, and 1,639 real pulsar examples.

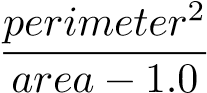
Attribute includes: 1) Mean of the integrated profile. 2) Standard deviation of the integrated profile. 3) Excess kurtosis of the integrated profile. 4) Skewness of the integrated profile. 5) Mean of the DM-SNR curve. 6) Standard deviation of the DM-SNR curve 7) Excess kurtosis of the DM-SNR curve 8) Skewness of the DM-SNR curve 9) Class (0 / 1)

*Alcohol QCM Sensor Dataset Data Set:* Five different QCM gas sensors are used, and five different gas measurements (1-octanol, 1-propanol, 2-butanol, 2-propanol and 1-isobutanol) are conducted in each of these sensors. In the dataset there are 5 types of dataset: QCM3, QCM6, QCM7, QCM10, QCM12. In each of dataset, There is alcohol classification of five types, 1-octanol, 1-propanol, 2-butanol, 2-propanol, 1-isobutanol.

The gas sample is passed through the sensor in five different concentrations. These concentrations are: Concentration Air ratio (ml) Gas ratio (ml) 1) 0*.*799 0*.*201 2) 0*.*700 0*.*300 3) 0*.*600 0*.*400 4) 0*.*501 0*.*499 5) 0*.*400 0*.*600. The last five columns are used to demonstrate the classification of each gas sample. 0 value refers to *false*. 1 value refers to *true*.

*Breast Cancer Wisconsin (Diagnostic) Data Set:* Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. Separating plane was obtained using Multisurface Method-Tree (MSM-T) [K. P. Bennett, ”Decision Tree Construction Via Linear Programming.” Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society, pp. 97-101, 1992], a classification method which uses linear programming to construct a decision tree. Relevant features were selected using an exhaustive search in the space of 1 − 4 features and 1 − 3 separating planes. The actual linear program used to obtain the separating plane in the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: ”Robust Linear Programming Discrimination of Two Linearly Inseparable Sets”, Optimization Methods and Software 1, 1992, 23-34].

Attribute includes: 1) ID number 2) Diagnosis (M = malignant, B = benign) 3-32) 3) radius (mean of distances from center to points on the perimeter) 4) texture (standard deviation of gray-scale values) 5) perimeter

6) area 7) smoothness (local variation in radius lengths) 8) compactness ( ) 9) concavity (severity of concave portions of the contour) 10) concave points (number of concave portions of the contour) 11) symmetry 12) fractal dimension (”coastline approximation” - 1)

*Cervical Cancer Behavior Risk Data Set:* The dataset contains 19 attributes regarding ca cervix behavior risk with class label is ca cervix with 1 and 0 as values which means the respondent with and without ca cervix, respectively.

Attribute includes: 1) behavior eating 2) behavior personalHygine 3) intention aggregation

4) intention commitment 5) attitude consistency 6) attitude spontaneity 7) norm significantPerson

8) norm fulfillment 9) perception vulnerability 10) perception severity 11) motivation strength

12) motivation willingness 13) socialSupport emotionality 14) socialSupport appreciation

15) socialSupport instrumental 16) empowerment knowledge 17) empowerment abilities

18) empowerment desires 19) ca cervix (this is class attribute, 1=has cervical cancer, 0=no cervical cancer)

*Reuter 50 50 Data Set:* The dataset is used for authorship identification in online Writeprint which is a new research field of pattern recognition. The dataset is the subset of *RCV* 1. These corpus has already been used in author identification experiments. In the top 50 authors (with respect to total size of articles) were selected. 50 authors of texts labeled with at least one subtopic of the class CCAT(corporate/industrial) were selected. That way, it is attempted to minimize the topic factor in distinguishing among the texts. The training corpus consists of 2*,*500 texts (50 per author) and the test corpus includes other 2*,*500 texts (50 per author) non-overlapping with the training texts.

This dataset contain only pure text files. Attributes of the dataset are character n-grams(n=1-5).

*24 Documents in Three Topics:* This dataset contains 24 text documents which belong to three topic folders: *C*1 airline safety, *C*4 hoof and mouth disease, and *C*7 mortgage rates. Each topic folder contains 8 documents.

This dataset contain only pure text files. Attributes of the dataset are character n-grams(n=1-5).

## **Preprocessing**

The program uses different ways and tools to preprocess real number datasets and text datasets.

**2.2.1 Real Number Datasets**

The program accepts real number datasets in *.csv*, *.data*, or *.txt* format. Delimiters are required and specified.

1. Input data files are read and stored by line. (In most cases, the first line is skipped, which are always table titles.)
2. Each line is splited into array (or list, or whatever suitable data structure) based on specified delimiters.
3. For datasets that have categories or class attributes, the column of category or class for each line, i.e., piece of data, is stored separately. This will be used for later performance evaluation.
4. For each piece of data, the rest of columns eare first parsed into real numbers, i.e., *double*, and then stored in array data structure.
5. All vectors (arrays) for all data points are stored in list data structure, and are now ready to serve as an input of the clustering algorithm.

**2.2.2 Text Datasets**

The program accepts text datasets in *.txt* format. Delimiter is not required. The program mainly uses **Stanford coreNLP** package to preprocess text documents. A fixed text file, **stopwords.txt**, is used in every preprocessing of datasets. This text file stores all stop words that would be removed in preprocessing part.

1. Each input data file is read and stored as a single string.
2. Stop words (specified in *stopwords.txt*) and all punctuation are removed.
3. The content string, now without stop words or punctuation, are tokenized (split by space).
4. Apply **Name Entity Recognition** feature in *Stanford coreNLP* package to detect name entities, and label them.
5. For words (tokens) that are not labeled as name entities, change them all to lower cases.
6. Apply **Lemmatizer** feature in *Stanford coreNLP* pacakge to lemmatize words that are not name entities.
7. Use self-implemented **2-gram** and **3-gram** to detect noun groups (such as “airline company”), and merge them as a single token.
8. After executing above to each document in the input dataset, calculate the **TFIDF matrix**. The calculated *TFIDF matrix* is ready to serve as input for clustering algorithm.

# METHODOLOGY

This section discusses the models and codes. In order to explain the modeling, some special behaviors and characteristics of dolphins are first presented.

## **Behaviors**

*Echolocation:* Dolphin has good eyesight, but the good eyesight is not quite helpful with predation in poor light conditions under the sea. Instead, the dolphin uses a special ability to search for prey, which is echolocation. In echolocating, it produces short broad-spectrum burst-pulses that sound to us like ”clicks.” These ”clicks” are reflected from objects of interest to the whale and provide information to the whale on food sources.With the help of echo, the dolphin can have a better perception of the surrounding environment.

**Cooperation and Information Exchange:** In most cases, predatory behavior is not achieved by one dolphin alone, but finished by cooperation. Facing large preys or swarms of preys, dolphins cooperate with each other and exchange their information using echolocation. Dolphins also divide their labor in many cases. For instance, the dolphins close to the prey are responsible for tracking the movements of the prey and the dolphins far away from the prey form a circle to surround the prey.

## **Predation Stages**

The algorithm modeling the clustering by simulating the predation activity of dolphins. The whole predation process can be seen as three phases: individual search, information exchange, and predation.

* *Individual Search:* each dolphin searches its best-fit prey within a given search radius. How ”suitable” a prey is to a dolphin is determined by a fitness function. And the optimal prey for each dolphin, if any, is stored.
* *Information Exchange:* dolphins exchange their information if necessary using echolocation. Specifically, one dolphin sends the information of its optimal prey to the other dolphin if 1) it currently finds a better prey and 2) the sound ”may” be transmitted to that dolphin before the end of the whole procedure. We use a matrix, i.e., two-dimensional array to keep track of the remaining time of sound to travel from one dolphin to another. When the sound arrives, the destination dolphin will make its own decision of whether it should keep its individual-searched prey or replaces by the prey found by neighbors.
* *Predation:* dolphins take action corresponding to the optimal prey, if any. And they moves to a certain position around their target preys, which depends on the distance to the target prey.

## **Models Construction**

Input training datasets are first pre-processed, as mentioned in Section 2.2. The document vector (raw data for real number datasets and TFIDF for text documents) is input to algorithm in **two-dimensional array**, i.e., matrix form. The algorithm is implemented using object oriented programming.

**3.3.1 Objects**

There are two main objects in this model: **Dolphin** and **Prey**. Both objects store the original document vector **.raw** (which is very possible to be high-dimensional) and the newly constructed vector in clustering space **.position**. The original document vector is used to computer the similarity between two objects, i.e. two data points, which is essentially used in clustering procedure. The newly constructed vector in clustering space is used for visualization.

Specifically, **Dolphin** has some extra attributes, which are its search radius **Dolphin.searchR** used in individual search phase and its optimal prey **Dolphin.optimal**. It also has attribute **Dolphin.cluster** to store data points that are assigned to its belonging cluster, which is modeled as preys being eaten by this dolphin. **Prey** has an extra attribute **Prey.status** to keep track of whether a prey is alive or eaten.

## **Similarity measure**

The project now implements 2 similarity measure: cosine similarity and Euclidean distance. (Since these two similarity measure are common, the report would not explain them in detail here.

The function is implemented in **Dolphin** class, requiring two objects: the original document vector of two measured objects.

**3.4.1 Fitness Function**

A special function needs to be specified here. Our algorithm uses a specific function to calculate “how suitable” a prey is to a dolphin. This function can be changed depending on features of input documents and can be improved by multiple times of testing. The function is implemented in **Dolphin** class, requiring two parameters: a **Dolphin** object and a **Prey** object. For now, in order to avoid ambiguity, the paper uses **fitness()** to refer to it.

In any case, the function should be related to similarity measure. In our case, the similarity measure is either cosine similarity or Euclidean distance. So far, two fitness functions are tested. Here use *D* to denote a **Dolphin** object and *P* a **Prey** object.

* *fitness*(*D,P*) = *sim*(*D.raw,P.raw*): Simply use similarity to determine how suitable a prey is to a dolphin. In this case, for cosine similarity measurement, the larger the fitness value is, the better the prey is; for Euclidean distance measurement, the smaller the fitness value is, the better the prey is.
* *fitness*(*D,P*) = *sim*(*D.raw,P.raw*) × *dense*: Here a new attribute **dense** is introduced. It measures the number of alive preys “nearby” *P*. More “nearby” alive preys there are, the better the *P* is to *D*.

**3.4.2 Initialization**

At the very first beginning, the algorithm converts all data points vectors to a two-dimensional vectors, which are used for visualization. This procedure can be done either randomly or principle component analysis. The current testing results are based on codes using random method.

The algorithm takes input from user (or programmer) an integer *k* to be the initial number of clusters. The algorithms then selects *k* data points randomly and treat them as **Dolphin**. The algorithm initializes the *k*

**Dolphin** and store them as [*D*1*,D*2*,...,Dk*]. The rest of the data points are treated as **Prey**, and are stored as

[*P*1*,P*2*,...,Pm*].

The *k* **Dolphin**s are treated as cluster centroids, and the rest of **Prey**s are data points needed to be assigned.

**3.4.3 Individual Search**

The algorithm then starts its clustering iteration. The first step is individual search.

In this stage, each **Dolphin** *Di* calculates the **fitness** values of **Prey** *Pj* such that *sim*(*Pj,Di*) *> Di.searchR*.

(Note that here *sim*() simply means using one similarity measure.) And find its optimal prey as

*Di.optimal* = argmin*fitness*(*Di,Pj*)

*Pj*

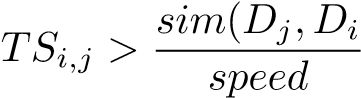
where *Pj sim*(*Pj,Di*) *> Di.searchR*.

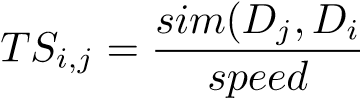
**3.4.4 Call**

After each **Dolphin** finishes its searching and saving its optimal prey if any, the algorithm enters the call phase.

In this phase, each **Dolphin** decides whether it is “needed” and “worthy” to send sound to neighbor **Dolphin** to inform them the information of its individual searched optimal prey. The time for sound to travel from one **Dolphin** to another needs to be recorded, and we use a *k* **TS** matrix to realize this. In **TS**, each entry *TSi,j* demonstrates the remaining time for sound to travel from *Dj* to *Di*. All entries are initialized as a very big number, and is updated as follows.

For each entry *TSi,j*, if

1. *fitness*(*Dj,Dj.optimal*) *< fitness*(*Di,Di.optimal*) and
2. ,

update.

The first condition checks if *Dj* has found a better prey than *Di* has in current iteration. The second condition checks if the current recorded time of sound to travel between *Dj* and *Di* is longer than re-sending the sound from *Dj* to *Di* given their current **position** in clustering space. If both conditions are satisfied, then *Dj* sends sound to *Di* and the entry *TSi,j* is updated. Otherwise, it is not need to exchange information and nothing is changed.

**3.4.5 Reception**

After every entry in **TS** is checked and updated if needed, the algorithm enters the reception phase.

In this phase, every entry in **TS** decreases by 1 to demonstrates the proceeding of iterations (which is time in real life). The algorithm then goes through every entry and checks if any sound is arrived. If *TSi,j <*= 0, sound sent from *Dj* arrives at *Di*, and *Di* now knows full information of *Dj*’s current optimal prey. Therefore it is time for *Di* to make a decision: whether to keep its individual searched prey or replaced the optimal solution by its newly received prey of its neighbor. If *fitness*(*Di,Di.optimal*) *> fitness*(*Dj,Dj.optimal*), this means if *Di* is willing to move closer to *Dj*, then it can get a better prey than it currently can. Therefore, the algorithm first **move** *Di* to *Dj.optimal* using a specific function rule which will be specified later. However, in clustering case, this also means that *Di* is actually not a very good centroid. To understand this, consider that for *Di* to take neighbor *Dj*’s optimal prey, there can be three possible reasons:

1) *Di* does not have very closed data points nearby; 2) *Di* does not have dense enough data points group around; 3) *Di* is not very far away from *Dj*.

The three possible reason are all clues for bad centroid choice. To fix this problem, our algorithm simply changes the identity of *Di* from **Dolphin** to **Prey**. The algorithm initializes a new **Prey** using the basic information of *Di*, remove *Di* from **Dolphin** array, and adds the new **Prey** object to **Prey** array. The algorithm then “releases” all preys previously eaten by *Di*. That is all data points previously assigned to *Di*’s cluster are now back to clustering space. The algorithm goes through *Pi* in *Di.cluster*, and sets *Pi.status* = ’a’. Reset the *i*th row and *i*th column in **TS**. The final thing needs to be checked is that based on this change, does *Dj*’s decision of optimal prey need to be changed as well? The algorithm checks: if *fitness*(*Dj,Dj.optimal*) *> fitness*(*Dj,Di*), this means the newly constructed **Prey**, which was **Dolphin** *Di* before, is actually a better choice for *Dj*. In this case, set *Dj.optimal* to be the newly constructed **Prey**. Otherwise, *Dj* still keeps its own individual searched prey and nothing needs to be changed here.

Otherwise, if *TSi,j >* 0, this simply means that the sound has not arrived yet. And therefore no action needs to be done here.

**3.4.6 Predation (Clustering)**

After every entry in **TS** is checked for 0, the algorithm enters the predation phase. This is the phase where data points assignment is done.

Before formally entering predation phase, a quick action needs to be done for the case when *Di.optimal* = *Dj.optimal*. That is, when two **Dolphin** now have the same optimal prey. To deal with this situation, the algorithm compares **fitness** values of these two **Dolphin**. If *fitness*(*Di,Di.optimal*) *< fitness*(*Dj,Dj.optimal*), then, clearly, *Di* is a better clustering centroid for the shared optimal prey. Therefore in this case, we first move *Dj* to a certain **position** depending on the **position** of the shared prey using a special **move** function. And then set *Dj.optimal* to be nothing. This means we still keep *Dj*’s identity as **Dolphin**, but nullify its choice of data point to be assigned with.

After checking above, the algorithm enters predation, i.e., clustering, phase formally. In this phase, each **Dolphin** “eats” its optimal prey if any. That is the algorithm assigns *Di.optimal* to *Di*’s cluster by *Di.cluster.add*(*Di.optimal*. The algorithm then **move** *Di* to a certain **position** around *Di.optimal*.

**3.4.7 Check Conditions**

After each data point assignments of each *Di*, the algorithm starts to check some conditions.

The first condition to be checked is whether it is time to stop the iteration, i.e., whether the clustering is finished. The algorithm goes through **Prey** list and checks if all *Pi.status*=‘d’. If so, all **Prey** are eaten which means all data points are assigned. The iteration stops and the algorithm returns the clustering results.

Otherwise, there are still alive **Prey** and there are still data points waiting to be assigned.

Another check needed is whether there are still *k* number of **Dolphin**. That is, whether new cluster centroid needs to be initialized. As mentioned in Section 4.3.6, some **Dolphin** may detected to be illed-chosen, and therefore are converted to **Prey** identity while clustering. In this situation, a new **Dolphin** needs t be initialized to guarantee the *k* number. Therefore, the algorithm picks a **Prey** among alive **Prey** randomly, and changes its identity from **Prey** to **Dolphin**. (Note that it is important to set this condition check after the condition check for iteration termination. If the initial choice of *k* is too big, i.e. no need for that many clusters, the redundant cluster centroids would be removed in Section 4.3.6 phase, and the algorithm would directly finish with that less but more suitable number of *k* when all data points are assigned.)

The algorithm then checks if *k* is enough. That is, if new clusters are needed for the datasets. The algorithm goes through all “alive” **Prey** and checks if ∃*Pi* s.t. *Pi.status*=‘a’ and *sim*(*Pi,Dj*) *< searchR* ∀*Dj*. If so, then there exists **Prey** that is not assigned but is far away from any of the existing clustering centroid. In this case, clearly, the algorithm needs to increment *k* and initializes a new cluster centroid to represent those far-away data. The algorithm picks randomly among those “isolated” **Prey**, and converts the picked one’s identity from **Prey** to **Dolphin**. Increment *k* and then continue the iteration.

The final action of the whole iteration is that at the end of every iteration, reset each existing **Dolphin**’s optimal prey. This is important since it is possible that in some iterations, some **Dolphin** does not find any **Prey** within their search radius. We definitely do not want them to do action to the optimal preys that found in previous iterations.

**3.4.8 Move Function**

As mentioned many times above, the algorithm has another important function implemented which is used to update **Dolphin**’s **position** based on its target **Prey**. The mathematical formula is borrowed from Wu’s paper.

[{citation}].

For a **Dolphin** *D* and its target **Prey** *P*, there are three different cases. In all three cases, a value *R*2 is first computed, and *D* is moved to some **position** that is *R*2 away from target **Prey**.

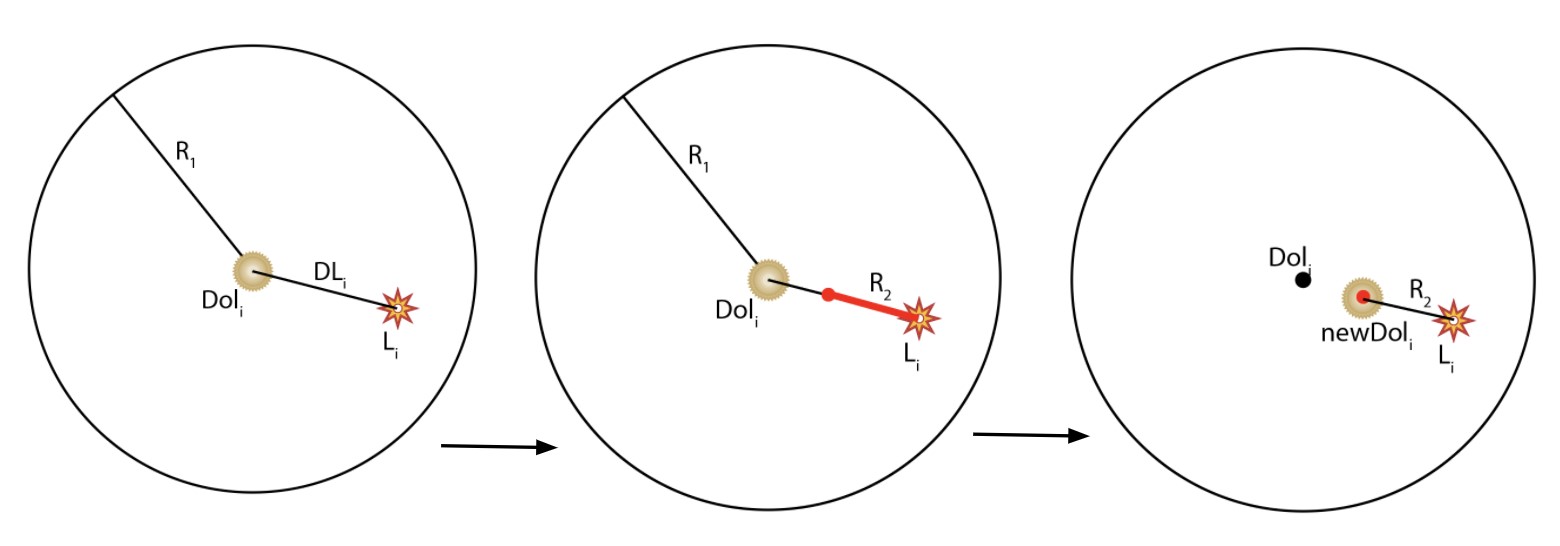


Figure 1. Case 1: *P* is within *D*’s search region

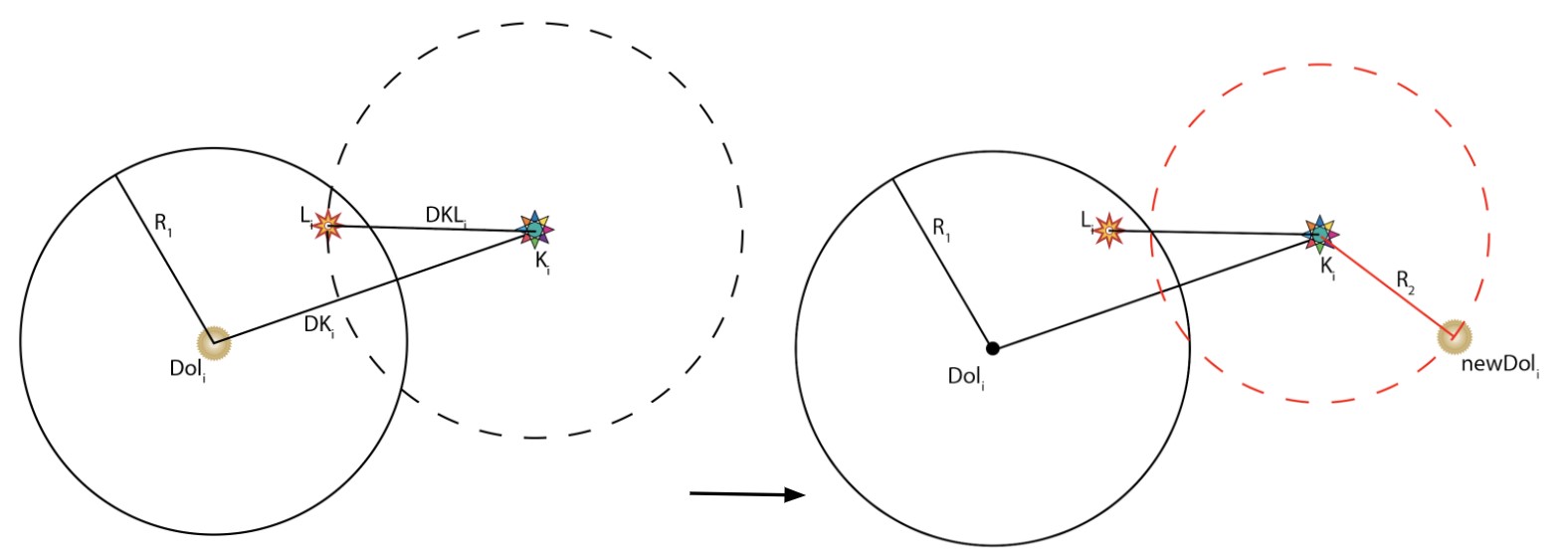
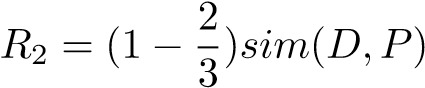
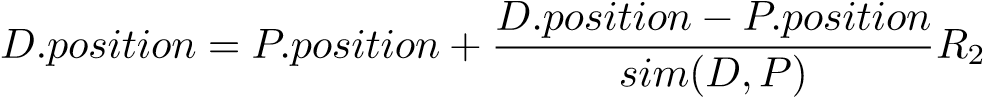


Figure 2. Case 2.1: *P* is closer to *P*’ than to *D*

**Case 1:** *P* **is within** *D***’s search region** In this case, compute *R*2 as



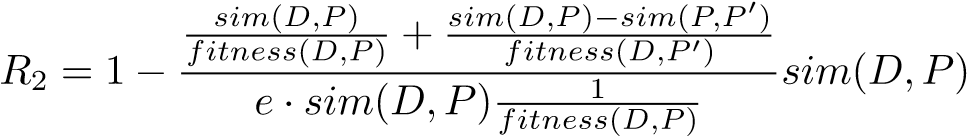
where *e* is a constant named the ‘radius reduction coefficient’, which is greater than two and usually set as three or four. *D* then is simply moved towards *P* in straight line, and gets its new **position** as



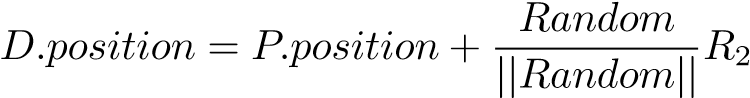
The move is demonstrated in Fig. 1.

**Case 2:** *P* **is out of** *D***’s search region** This means *D* must get *P* from its neighbor. Let *P*0 denote *D.optimal*, which is *D*’s individual searched **Prey**. And we have two sub-cases.

**Case 2.1:** *P* **is closer to** *P***’ than to** *D* This means compared to *D*, *P* is closer to *D*’s individual searched **Prey**. In this case, compute *R*2 as



Different from Case 1, now instead of moving *D* to *P* in line, the algorithm moves *D* to a random position that is *R*2 around *P*, which is along a circle centered at *P* with radius *R*2. And we compute *D*’s new **position** as



The move is demonstrated in Fig. 2.

**Case 2.2:** *P* **is closer to** *D* **than to** *P*0 This means compared to *D*’s individual searched **Prey**, *P* is closer to *D*. In this case, compute *R*2 as

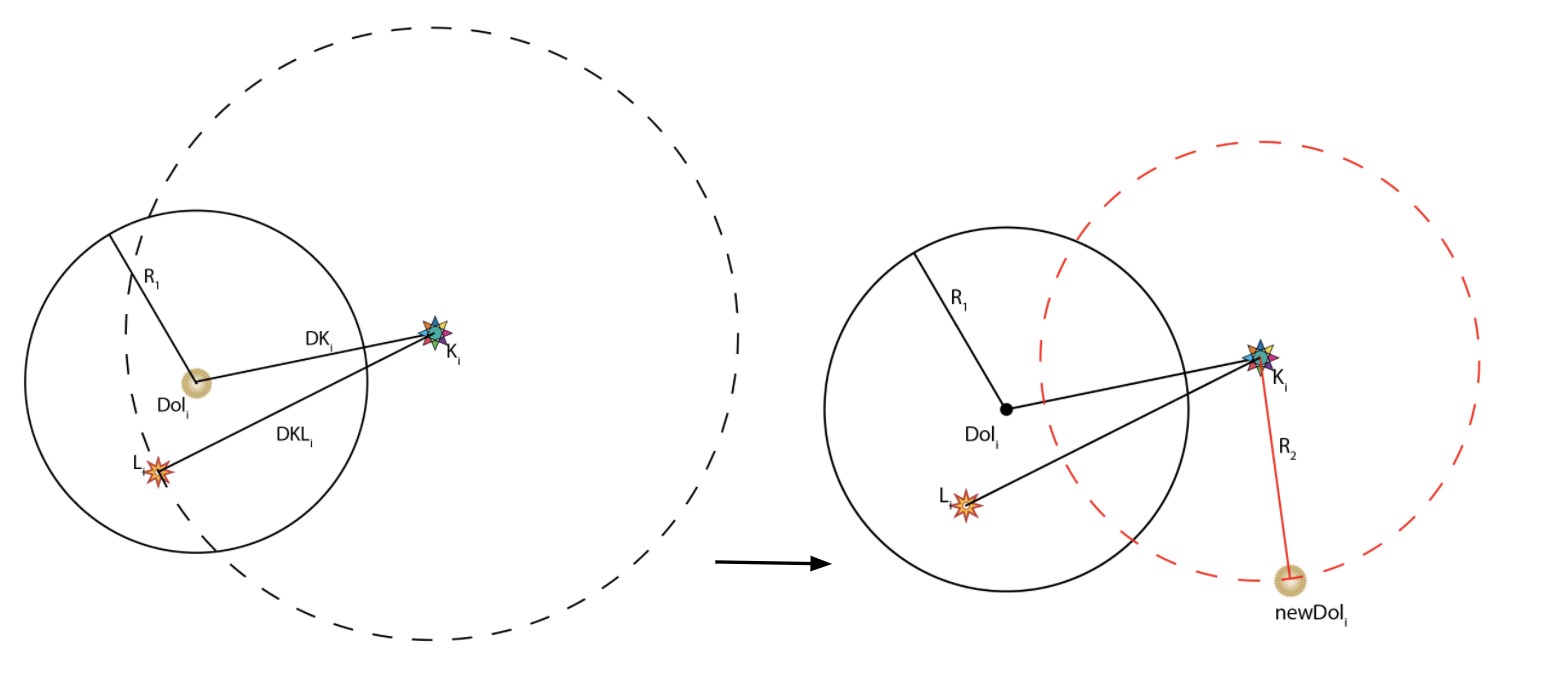
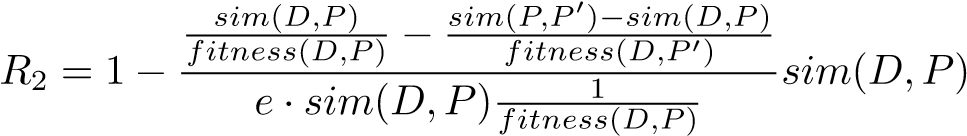
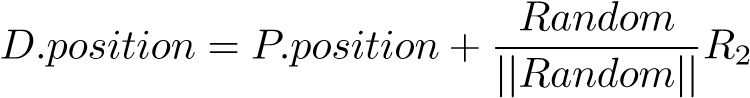


Figure 3. Case 2.2: *P* is closer to *D* than to *P*0



Again, now instead of moving *D* to *P* in line, the algorithm moves *D* to a random position that is *R*2 around *P*, which is along a circle centered at *P* with radius *R*2. And we compute *D*’s new **position** as



The move is demonstrated in Fig. 3.

## **Algorithms**

Pseodu-codes are provided in **Algorithm 1-6**.

Note that the **move()** algorithm would not be provided here since it has already been explained quite in detail in Section 3.4.8. A variable *c* is used as an extra parameter for the function in program in order to specify whether the dolphin is moving to a prey that is found by itself or found by other dolphins. **i** denotes individual searched prey and **n** denotes neighbor searched prey.

**3.5.1 Dolphin Class**

* position: double[]
* raw: double[]
* searchR:
* cluster: ArrayList¡Prey¿
* optimal: Prey
* name: String (used for text documents datasets)

**3.5.2 Prey class**

* position: double[]
* raw: double[]
* status: char
* name: String (used for text documents datasets)

## **Cluster class**

* rawArr: double[][]
* docs: ArrayList¡Document¿ (used for text documents datasets)
* k: int
* searchR: double
* speed: double
* e: int

**Algorithm 1** Search (for cosine similarity measure)

**Implemented in** Dolphin **class** double *maxFit* ← −1 int *found* = 0 **for** *Preyp* in *preyArr* **do**

**if** cosineSim(*this*.raw, *p*.raw)*> searchR* and *p*.status ! = ’d’ **then**

double *currFit* ← fitness(*this*, *p*, *preyArr*) **if** *currFit > maxFit* **then** *maxFit* ← *currFit this*.optimal←pfound← 1 **end if**

**end if if** *found* = 0 **then**

*this.*optimal ← new *Prey*() *this.*optimal.status ← ’d’

**end if**

**end for**=0

**Algorithm 2** Initialization

initialize a list **dolphinArr** to store **Dolphin** objects initialize a list **preyArr** to store **Prey** objects int[] *dolIdx* ← new int[*k*] *dolIdx* ← *k* randomly selected indexes within *rawArr* **for** int *i* ← 0, *i < rawArr.*length, *i* + + **do if** *i* in *dolIdx* **then**

double[] *newPos* ←= new double[2] *newPos* ← randomly generated 2−dimensional vectors *Dolphin newD* ← new *Dolphin*(*newPos,rawArr*[*i*]*,searchR*) *dolArr*.add(*newD*)

**else**

double[] *newPos* ←= new double[2] *newPos* ← randomly generated 2−dimensional vectors *Prey newP* ← new *Dolphin*(*newPos,rawArr*[*i*]) *preyArr*.add(*newP*)

**end if end for**=0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm 3** Call Phase |  |  |  |  |
| double[] *TS* ← new double[*dolphinArr*.length][*dolphinArr*.length] initialize all *TS* entries to be 10000000 **for** int *i* ← 0, *i < dolphinArr.*length, *i* + + **do**  **for** int *j* ← 0, *j < dolphinArr*.length, *j* + + **do**  *Dolphin Di* ← *dolphinArr*.get(*i*) *Dolphin Dj* ← *dolphinArr*.get(*j*) **if** *Dj*.optimal.status != ’d’ and *Di*.optimal.status != ’d’ **then if** *Dj*.fitness(*Dj*, *Dj*.optimal, *preyArr*) *> Di.*fitness(*Di,Di*.optimal, | *preyArr* | and | *TS*[*i*][*j*] | *>* |
| **then**  **end if else if** *Dj*.optimal.status = ’d’ and *Di*.optimal.status != ’d’ **then if** *Dj*.fitness(*Dj*, *Dj*.optimal, *preyArr*) *> Di.*fitness(*Di,Di*.optimal,  **then**  **end if**  **end if**  **end if**  **end for end for**=0 | *preyArr* and *TS*[*i*][*j*] | | | *>* |

# EVALUATIONS

This section display the clustering results of all test datasets described in Section 2.1. Performances for each datasets are evaluated, and clustering results visualizations are displayed.

*Important Note:*

* Like many other clustering algorithms, this dolphin swarm clustering algorithm’s result is different for every run. For this project, multiple runs have been attempted. The following performance measures use the best-performed results.
* For this algorithm, there are 3 adjustable parameters: **e**, **speed**, and *search radius*. The clustering results and performances vary by changing these 3 parameters. Multiple runs and attempts have been done to obtain the best-performed parameter values, and the following performance measures use the best-performed values.

## **Performance**

Intentionally, I have chosen datasets that have predictable category / class. The algorithm prints out the assigned cluster for each data points (as explained in detail in **README.txt**). The category / class / label of each cluster is determined by the majority.

Performance of each datasets are evaluated by measuring

* Confusion matrix
* Accuracy
* Precision for each attribute

**Algorithm 4** Reception Phase

**Step 2.3: Reception** initialize a list **removeDol** to store indexes of dolphins that are needed to be removed **for** int *i* ← 0, *i < dolphinArr.*length, *i* + + **do**

**for** int *j* ← 0, *j < dolphinArr*.length, *i* + + **do**

*Dolphin Di* ← *dolphinArr*.get(*i*)

*Dolphin Dj* ← *dolphinArr*.get(*j*) *TS*[*i*][*j*] ← *TS*[*i*][*j*] − 1 **if** *TS*[*i*][*j*] ≤ 0 **then** *TS*[*i*][*j*] ← 10000000 **if** *Dj*.fitness(*Dj*, *Dj*.optimal, *preyArr*) ≥ *Di*.fitness(*Di*, *Di*.optimal, *preyArr*) **then**

char *c* ← ’n’ **for** *Prey prey* in *Di*.cluster **do**

*prey*.status← ’a’

**end for**

*Di*.move(*Dj*.optimal, *e*, *c*, *preyArr*) re-initialize the *i*th row and *i*th columns in *TS Prey newPrey* ← new Prey(*Di*.position, *Di*.raw) *preyArr*.add(*newPrey*) *removeDol*.add(*i*)

**if** *Dj*.fitness(*Dj*, *Dj*.optimal) *< Dj*.fitness(*Dj*, *newPrey*) **then**

*Dj*.optimal← *newPrey* **end if**

**end if**

**end if end for**

**end for for** *Dolphin D* in *removeDol* **do**

*removeDol*.remove(*D*)

**end for**=0

**Algorithm 5** Predation Phase (Clustering Phase)

**for** *Dolphin D* in *dolphinArr* **do**

**if** *D*.optimal.status ! = ’d’ **then**

*D*.optimal.status ← ’d’ *D*.cluster.add(*D*.optimal) char *c* **if** cosineSim(*D*.optimal.raw, *D*.raw *> searchR* **then**

*c* ← ’i’ **else**

*c* ← ’n’

**end if**

*D*.move(*D*.optimal, *e*, *c*) **end if end for**=0

**Algorithm 6** Dolphin Swarm Clustering Algorithm

**while** true **do Step 1: Initialization** apply **Algorithm 2 Step 2.1: Search for** *Dolphin dolphin* in *dolArr* **do** *dolphin*.search() **end for Step 2.2: Call** apply **Algorithm 3 Step 2.3: Reception** apply **Algorithm 4**

**Special case: when two dolphins find the same preys**

initialize a hashmap (dictionary) **repeat** with key referring prey and value referring dolphin **for** *Dolphin dol* in *dolphinArr* **do**

**if** *dol*.optimal is an existing key in *repeat* **then**

*Prey p* ← *dol*.optimal

*Dolphin D*1 ← *repeat*.get(*p*) **if** *D*1.fitness(*D*1, *p*) ¿ *dol*.fitness(*dol*, *p*) **then**

char *c* **if** cosineSim(*dol*, *p > searchR* **then**

*c* ← ’i’ **else**

*c* ← ’n’

**end if** *dol*.move(*p*, *e*, *c*, *preyArr*) re-initialize *dol*.optimal to be NONE

**else**

move *D*1 and re-initialize *D*1.optimal to be NONE change *p*’s value in *repeat* to be *dol*

**end if**

**end if**

**end for Check Conditions** int *aliveNum* ← 0 **for** *Prey prey* in *preyArr* **do**

**if** *prey*.status = ’a’ **then**

*aliveNum* ← *aliveNum* + 1

**end if**

**end for if** *aliveNum* = 0 **then**

**break**

**end if if** *dolphin*.size *< k* **then** re-initialize a new *Dolphin newD* among alive *Prey*s *dolphinArr*.add(*newD*)

**end if**

**if** ∃ alive preys that cannot be reached by any existing dolphins **then**

initialize a new dolphin among those inaccessible preys store the new dolphin to *dolphinArr*

**end if**

**end while**

=0

Table 1. Confusion Matrix of **Iris Dataset**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Predicted Iris-Setosa | Predicted Iris-Virginica | Predicted Iris-Versicolor |
| Actual Iris-Setosa | 50 | 0 | 0 |
| Actual Iris-Virginica | 0 | 50 | 0 |
| Actual Iris-Versicolor | 0 | 1 | 49 |

Table 2. Performance Details of **Iris Dataset**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Iris-Setosa | Iris-Virginica | Iris-Versicolor |
| Precision | 1.0 | 0.98 | 0.98 |
| Recall | 1.0 | 1.0 | 0.98 |
| Accuracy |  | 0.99 |  |

• Recall for each attribute

Above performance measurements are displayed in Table 1-22. Note that for *HTRU* and *Reuter 50 50* dataset, partial of datasets were tested.

## **Clustering Results Visualization**

In order to visualize the data, this project scattered the 2-dimensional vectors inside of the clustering space, instead of the original multi-dimensional vectors, when the clustering process is finished. As described in Section 3, these 2-dimensional vectors inside of the clustering space are generated randomly at the very beginning, but would change (as *Dolphin* object *move* toward *Prey* object) during the clustering process. (More details about the *move* can be found in Section 3.4.8.)

This section provides plots for 1) the initial randomly generated 2-dimensional vectors of all data points and 2) the clustering results, with each cluster differing from each other by color or symbol, using the 2−dimensional vectors at the end of the clustering process.

Note that for the *Alcohol QCM* dataset, for save of space, only *QCM3*, *QCM6*, and *QCM* (all *QCM* (files summarized in one file) are plotted.

Results are shown in Fig 4-21.

# DISCUSSION

From the confusion matrices and performance measures displayed in Section 4.2, we can see that in all this dolphin swarm clustering algorithm is performing well. From Table 7 and Table 8, we can see that the performance on *Cervical Cancer Dataset* is not very well. This is because in this dataset, the distribution of two classes (0 and 1) are not very even. Similar situation happens to **Breast Cancer Dataset**. And from Table 5 and Table 6, we can see that the performance on this dataset is better than *Cervical Cancer* but still not as good as the other one. From this we can deduce this algorithm works better in dataset that data are distributed evenly among categories.

Another observation is that from Table 20 and Table 21, we may find the performance on *QCM* dataset, which is a summarization of all data from *QCM3, 6, 7, 10, 12*, is not very good. This is because as specifed in the dataset description in previous Sections, data from *QCM3, 6, 7, 10, 12* are tested categories using different techniques. Therefore, this may cause the dependency of testing results different among these 5 datasets. Therefore, when combining them together and attempting to predict categories for all, the performance is worse than clustering and predicting them separately.

Table 3. Confusion Matrix of **Documents Topics**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Predicted C1 | Predicted C4 | Predicted C7 |
| Actual C1 (Airline Safety) | 5 | 1 | 2 |
| Actual C4 (Hoof and Mouth Disease) | 0 | 8 | 0 |
| Actual C7 (Mortgage Rage) | 0 | 0 | 8 |

## Table 4. Performance Details of **Documents Topics**

|  |  |  |  |
| --- | --- | --- | --- |
|  | C1 (Airline Safety) | C4 (Hoof and Mouth Disease) | C7 (Mortgage Rage) |
| Precision | 1.0 | 0.89 | 0.8 |
| Recall | 0.625 | 1.0 | 1.0 |
| Accuracy |  | 0.875 |  |

Table 5. Confusion Matrix of **Breast Cancer**

|  |  |  |
| --- | --- | --- |
|  | Predicted Malignant | Predicted Benign |
| Actual Malignant | 195 | 17 |
| Actual Benign | 49 | 308 |

Table 6. Performance Details of **Breast Cancer**

|  |  |  |
| --- | --- | --- |
|  | Malignant | Benign |
| Precision | 0.80 | 0.95 |
| Recall | 0.92 | 0.86 |
| Accuracy | 0.88 |  |

Table 7. Confusion Matrix of **Cervical Cancer**

|  |  |  |
| --- | --- | --- |
|  | Predicted No cancer | Predicted Has cancer |
| Actual No cancer | 36 | 15 |
| Actual has cancer | 1 | 20 |

|  |  |  |
| --- | --- | --- |
|  | No cancer | Has Cancer |
| Precision | 0.97 | 0.95 |
| Recall | 0.71 | 0.95 |
| Accuracy | 0 | .76 |

Table 8. Performance Details of **Cervical Cancer** Table 9. Confusion Matrix of **QCM3 Dataset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Predicted 1-Oct | Predicted 1-Pro | Predicted 2-But | Predicted 2-pro | Predicted 2-iso |
| Actual 1-Octanol | 5 | 0 | 0 | 0 | 0 |
| Actual 1-Propanol | 0 | 5 | 0 | 0 | 0 |
| Actual 2-Butanol | 0 | 0 | 5 | 0 | 0 |
| Actual 2-Propanol | 1 | 0 | 0 | 4 | 0 |
| Actual 2-Isobutanol | 0 | 0 | 0 | 0 | 5 |

## Table 10. Performance Details of **QCM3 Dataset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1-Octanol | 1-Propanol | 2-Butanol | 2-Propanol | 2-Isobutanol |
| Precision | 0.83 | 1.0 | 1.0 | 1.0 | 1.0 |
| Recall | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| Accuracy |  |  | 0.96 |  |  |

Table 11. Confusion Matrix of **QCM6 Dataset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Predicted 1-Oct | Predicted 1-Pro | Predicted 2-But | Predicted 2-pro | Predicted 2-iso |
| Actual 1-Octanol | 5 | 0 | 0 | 0 | 0 |
| Actual 1-Propanol | 0 | 5 | 0 | 0 | 0 |
| Actual 2-Butanol | 0 | 0 | 5 | 0 | 0 |
| Actual 2-Propanol | 0 | 0 | 0 | 5 | 0 |
| Actual 2-Isobutanol | 0 | 0 | 0 | 0 | 5 |

## Table 12. Performance Details of **QCM6 Dataset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1-Octanol | 1-Propanol | 2-Butanol | 2-Propanol | 2-Isobutanol |
| Precision | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| Recall | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| Accuracy |  |  | 1.0 |  |  |

## Table 13. Confusion Matrix of **QCM7 Dataset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Predicted 1-Oct | Predicted 1-Pro | Predicted 2-But | Predicted 2-pro | Predicted 2-iso |
| Actual 1-Octanol | 5 | 0 | 0 | 0 | 0 |
| Actual 1-Propanol | 0 | 5 | 0 | 0 | 0 |
| Actual 2-Butanol | 0 | 0 | 5 | 0 | 0 |
| Actual 2-Propanol | 0 | 0 | 0 | 5 | 0 |
| Actual 2-Isobutanol | 0 | 0 | 0 | 0 | 5 |

## Table 14. Performance Details of **QCM7 Dataset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1-Octanol | 1-Propanol | 2-Butanol | 2-Propanol | 2-Isobutanol |
| Precision | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| Recall | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| hline Accuracy |  |  | 1.0 |  |  |

## Table 15. Confusion Matrix of **QCM10 Dataset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Predicted 1-Oct | Predicted 1-Pro | Predicted 2-But | Predicted 2-pro | Predicted 2-iso |
| Actual 1-Octanol | 5 | 0 | 0 | 0 | 0 |
| Actual 1-Propanol | 0 | 5 | 0 | 0 | 0 |
| Actual 2-Butanol | 0 | 0 | 5 | 0 | 0 |
| Actual 2-Propanol | 0 | 0 | 0 | 5 | 0 |
| Actual 2-Isobutanol | 0 | 0 | 0 | 0 | 5 |

## Table 16. Performance Details of **QCM10 Dataset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1-Octanol | 1-Propanol | 2-Butanol | 2-Propanol | 2-Isobutanol |
| Precision | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| Recall | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 |
| Accuracy |  |  | 1.0 |  |  |

## Table 17. Confusion Matrix of **QCM12 Dataset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Predicted 1-Oct | Predicted 1-Pro | Predicted 2-But | Predicted 2-pro | Predicted 2-iso |
| Actual 1-Octanol | 5 | 0 | 0 | 0 | 0 |
| Actual 1-Propanol | 0 | 3 | 0 | 0 | 2 |
| Actual 2-Butanol | 0 | 1 | 4 | 0 | 0 |
| Actual 2-Propanol | 0 | 0 | 0 | 3 | 0 |
| Actual 2-Isobutanol | 0 | 0 | 0 | 2 | 5 |

## Table 18. Performance Details of **QCM12 Dataset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1-Octanol | 1-Propanol | 2-Butanol | 2-Propanol | 2-Isobutanol |
| Precision | 1.0 | 0.75 | 1.0 | 0.6 | 0.71 |
| Recall | 1.0 | 0.6 | 0.8 | 1.0 | 0.71 |
| Accuracy |  |  | 0.8 |  |  |

Table 19. Confusion Matrix of **QCM Dataset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Predicted 1-Oct | Predicted 1-Pro | Predicted 2-But | Predicted 2-pro | Predicted 2-iso |
| Actual 1-Octanol | 21 | 3 | 0 | 0 | 0 |
| Actual 1-Propanol | 0 | 6 | 4 | 5 | 10 |
| Actual 2-Butanol | 0 | 5 | 11 | 5 | 4 |
| Actual 2-Propanol | 2 | 9 | 0 | 14 | 0 |
| Actual 2-Isobutanol | 2 | 5 | 1 | 5 | 14 |

## Table 20. Performance Details of **QCM Dataset**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1-Octanol | 1-Propanol | 2-Butanol | 2-Propanol | 2-Isobutanol |
| Precision | 0.84 | 0.21 | 0.69 | 0.47 | 0.5 |
| Recall | 0.84 | 0.24 | 0.44 | 0.56 | 0.52 |
| Accuracy |  |  | 0.52 |  |  |

## Table 21. Confusion Matrix of **HTRU Dataset**

|  |  |  |
| --- | --- | --- |
|  | Predicted Class 0 | Predicted Class 1 |
| Actual Class 0 | 1718 | 117 |
| Actual Class 1 | 44 | 120 |

## Table 22. Performance Details of **HTRU Dataset**

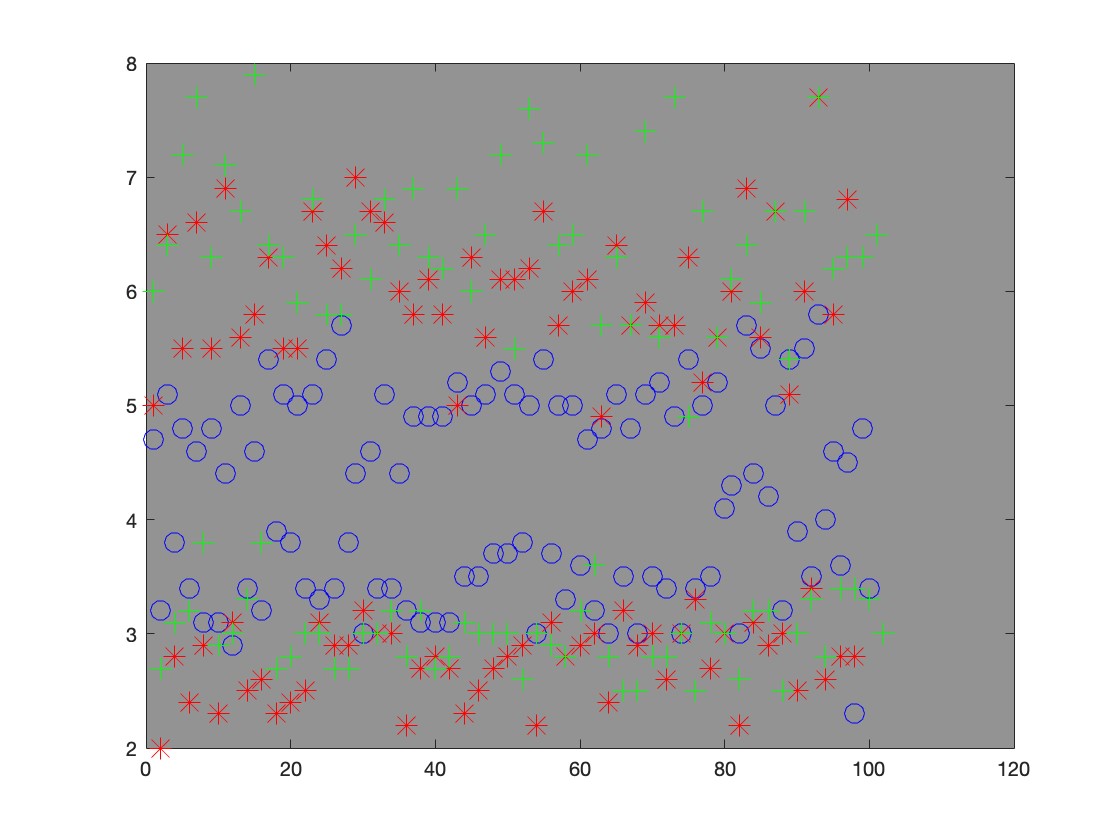
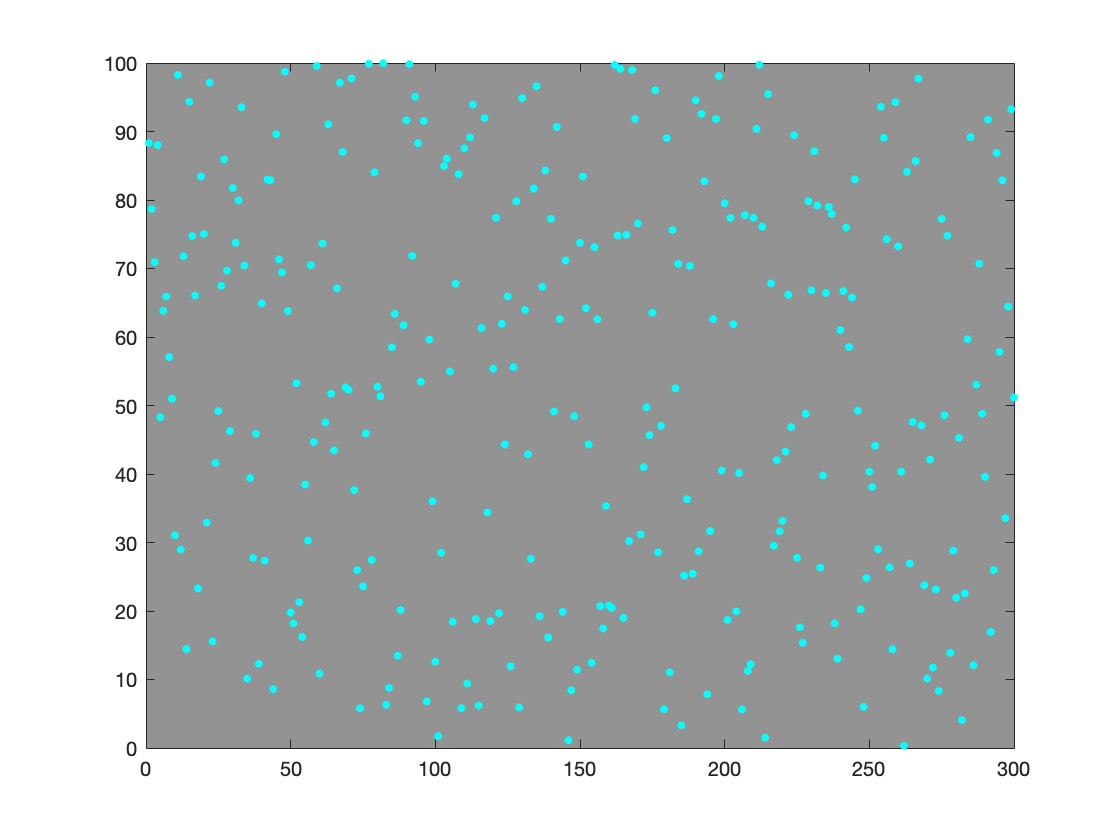
|  |  |  |
| --- | --- | --- |
|  | Class 0 | Class 1 |
| Precision | 0.98 | 073 |
| Recall | 0.94 | 0.73 |
| Accuracy | 0.92 | |

## Table 23. Confusion Matrix of **Reuter 50 50 Dataset**

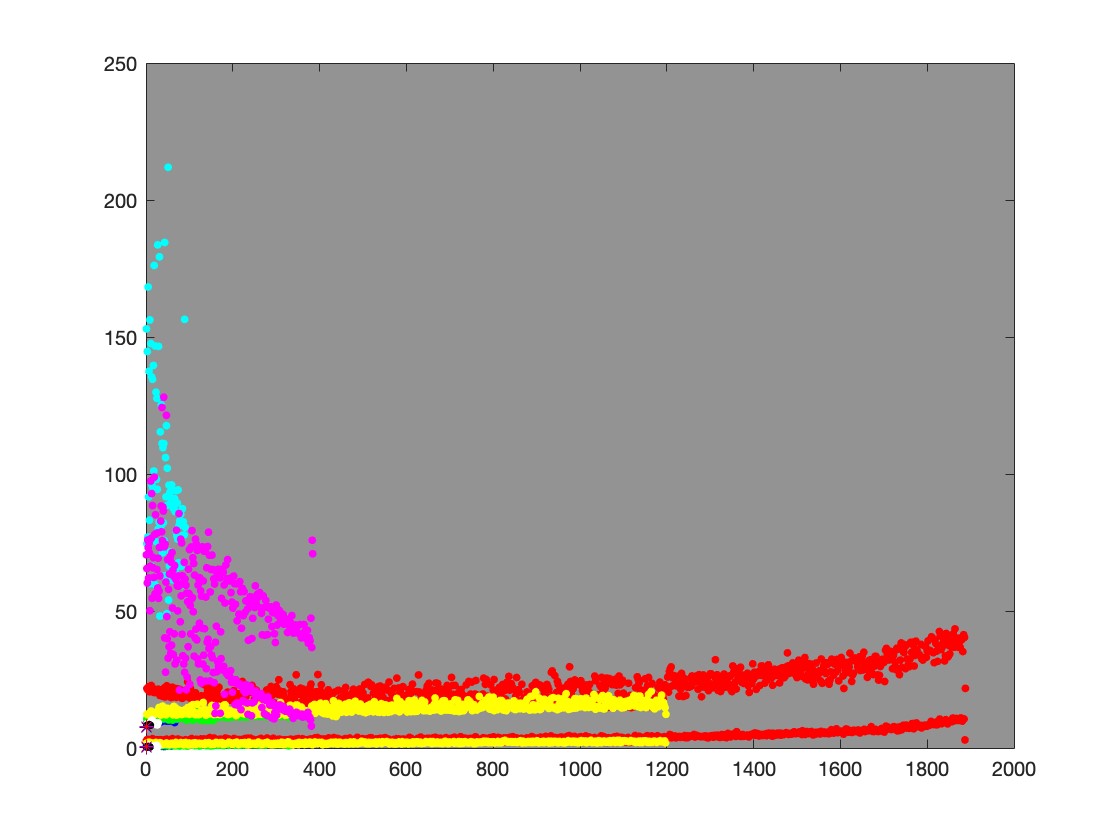
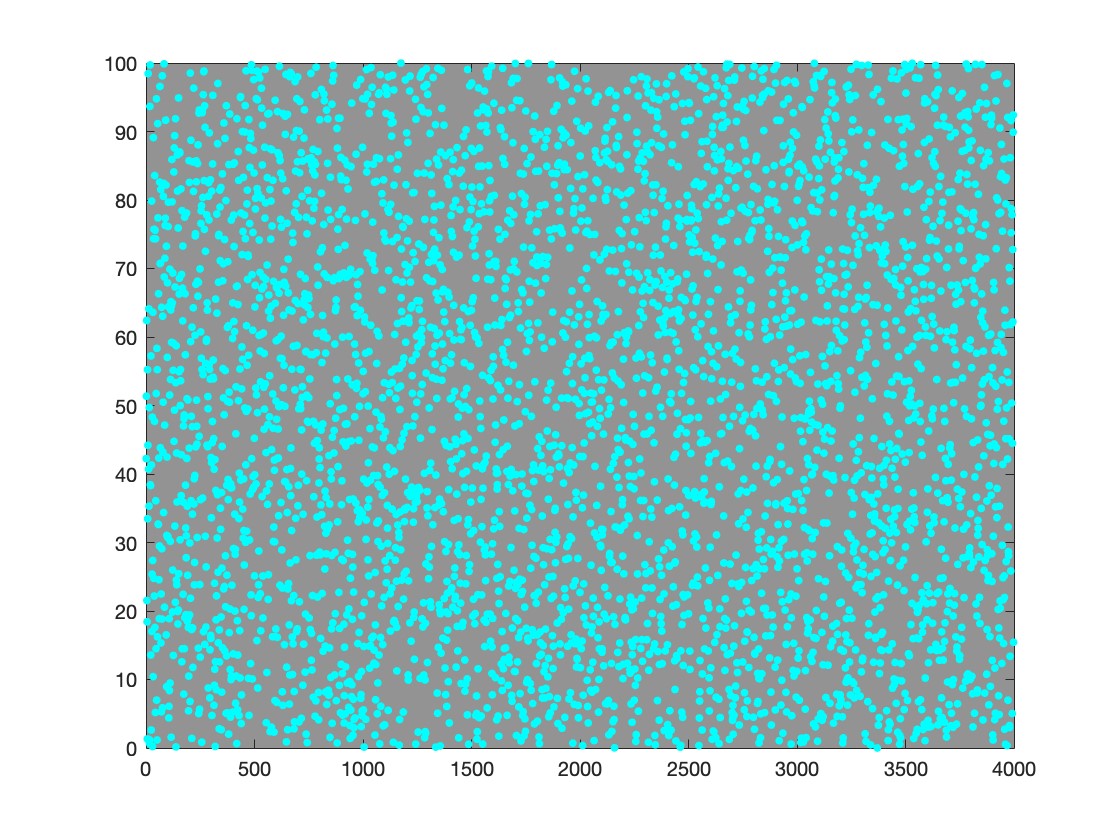
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Predicted PA | Predicted CA | Predicted HB | Predicted KB | Predicted SA |
| Actual Aaron Pressman | 11 | 0 | 0 | 1 | 0 |
| Actual Alan Crosby | 1 | 14 | 0 | 0 | 0 |
| Actual Bernard Hickey | 1 | 0 | 12 | 0 | 2 |
| Actual Benjamin Kang Lim | 0 | 0 | 1 | 13 | 1 |
| Actual Alexander Smith | 3 | 1 | 3 | 0 | 8 |

## Table 24. Performance Details of **Reuter**5050*Dataset*

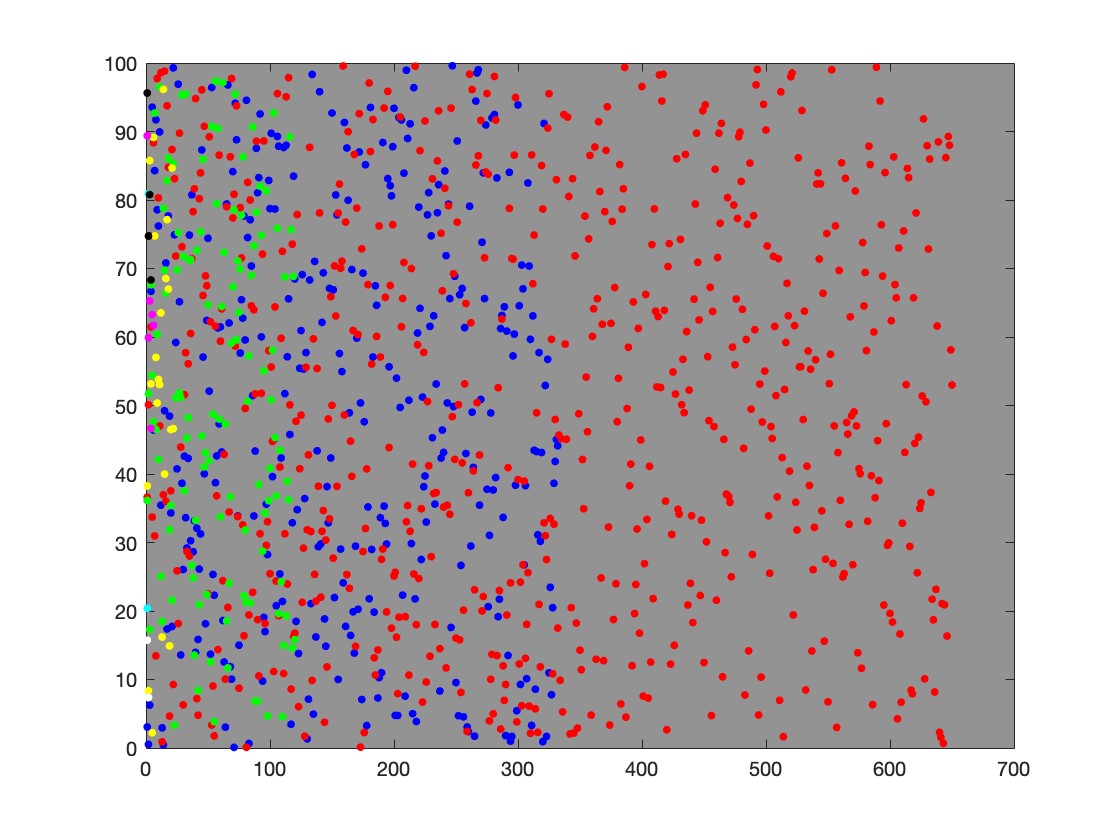
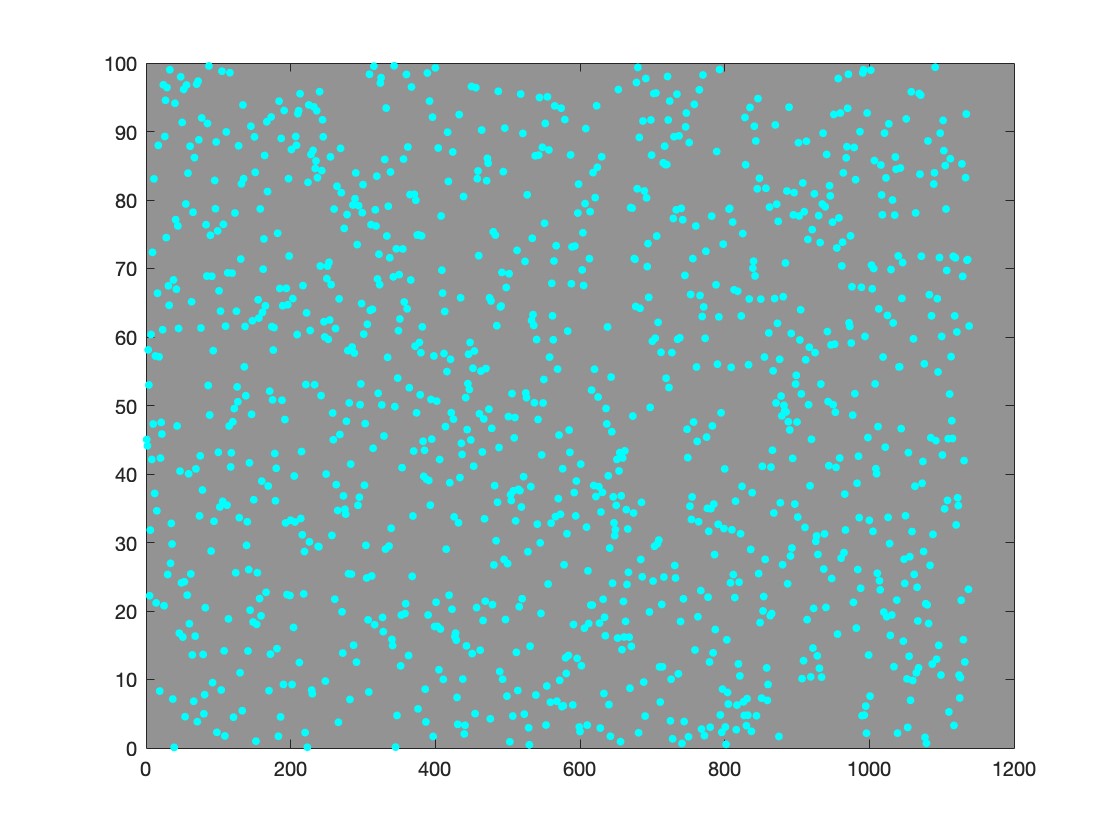
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Aaron Pressman | Alan Crosby | Bernard Hickey | Benjamin Kang Lim | Alexander Smith |
| Precision | 0.69 | 0.93 | 0.75 | 0.93 | 0.57 |
| Recall | 0.73 | 0.93 | 0.8 | 0.87 | 0.53 |
| Accuracy |  |  | 0.77 |  |  |



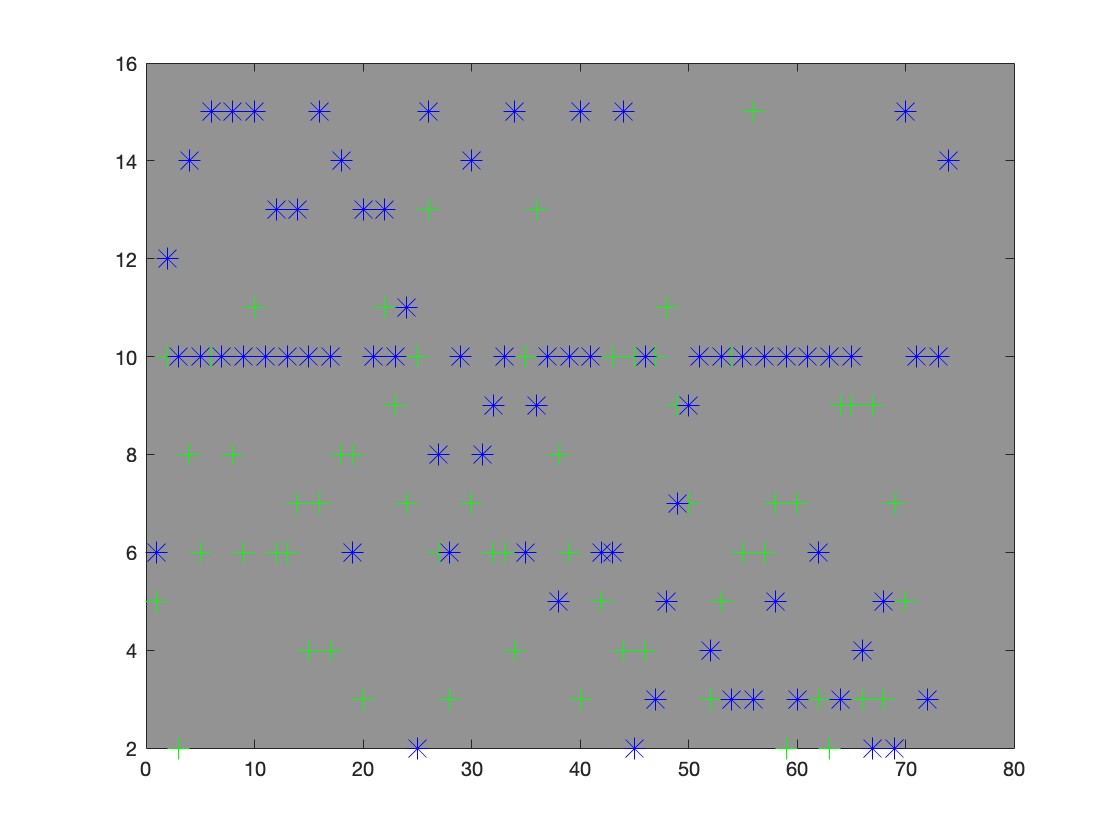
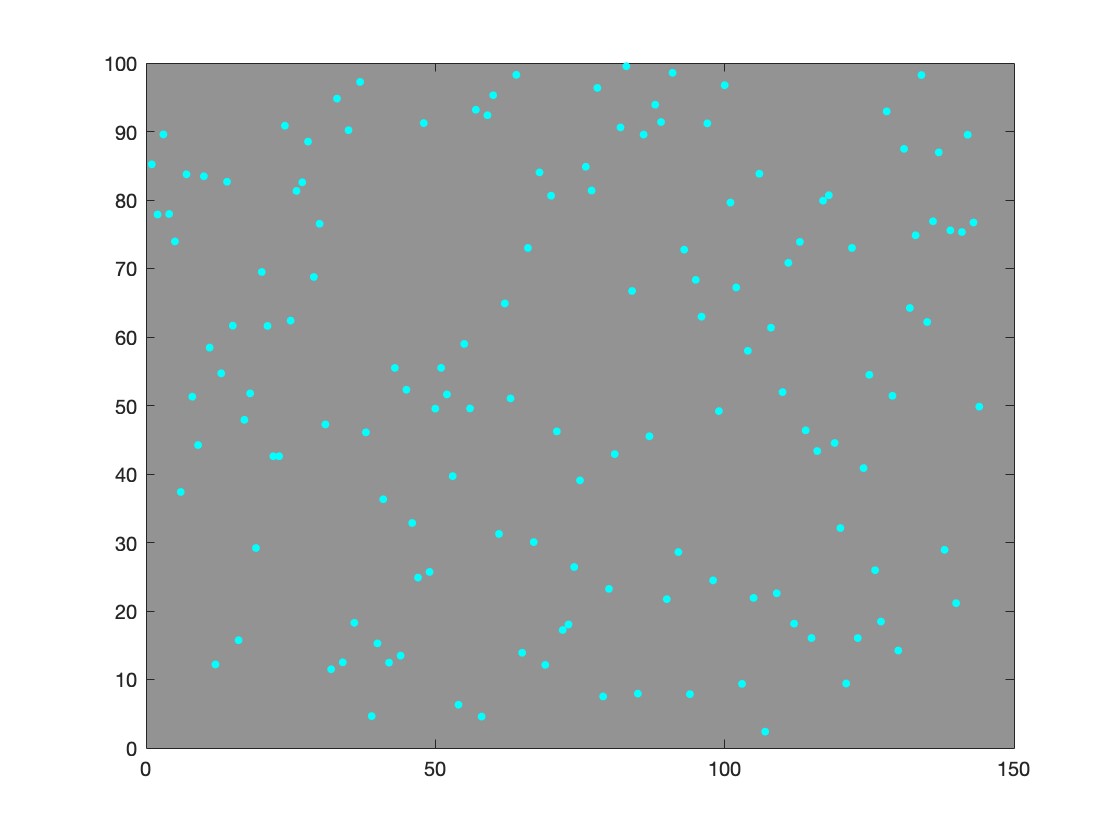
## Figure 4. Raw plots for **Iris Dataset** Figure 5. Clustering results for **Iris Dataset**



## Figure 6. Raw plots for **HTRU Dataset** Figure 7. Clustering results for **HTRU Dataset**

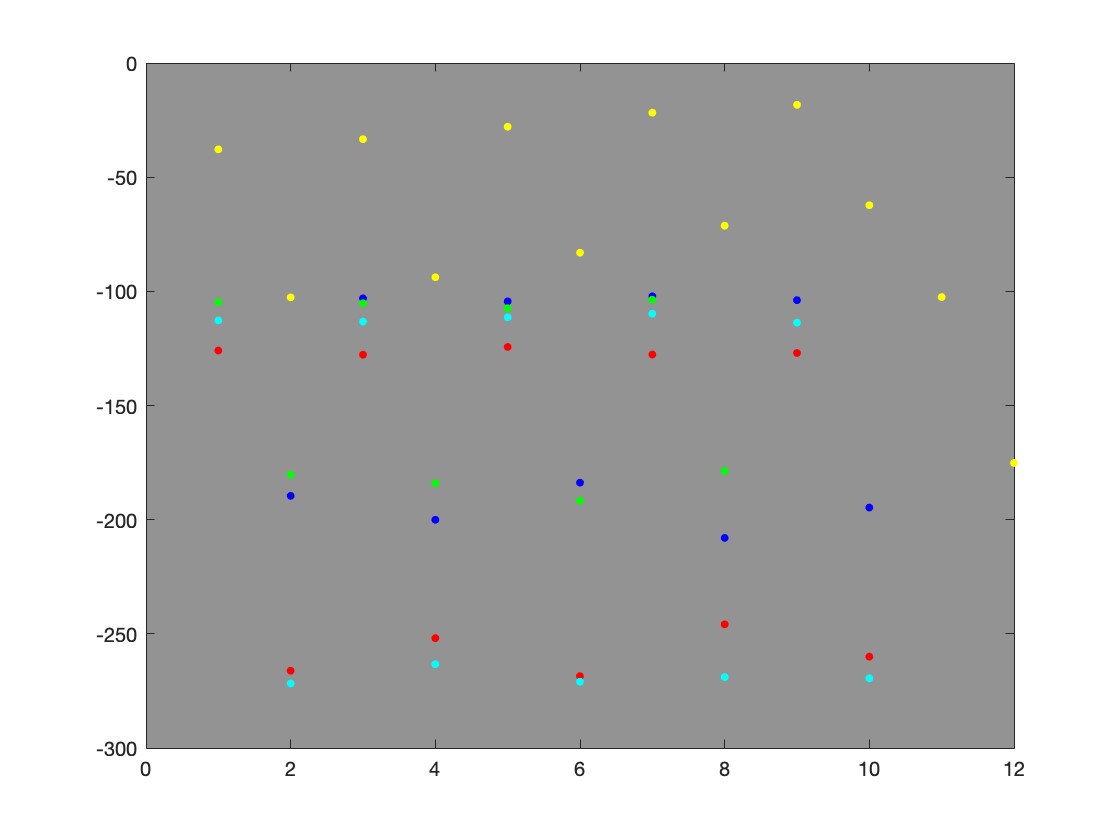
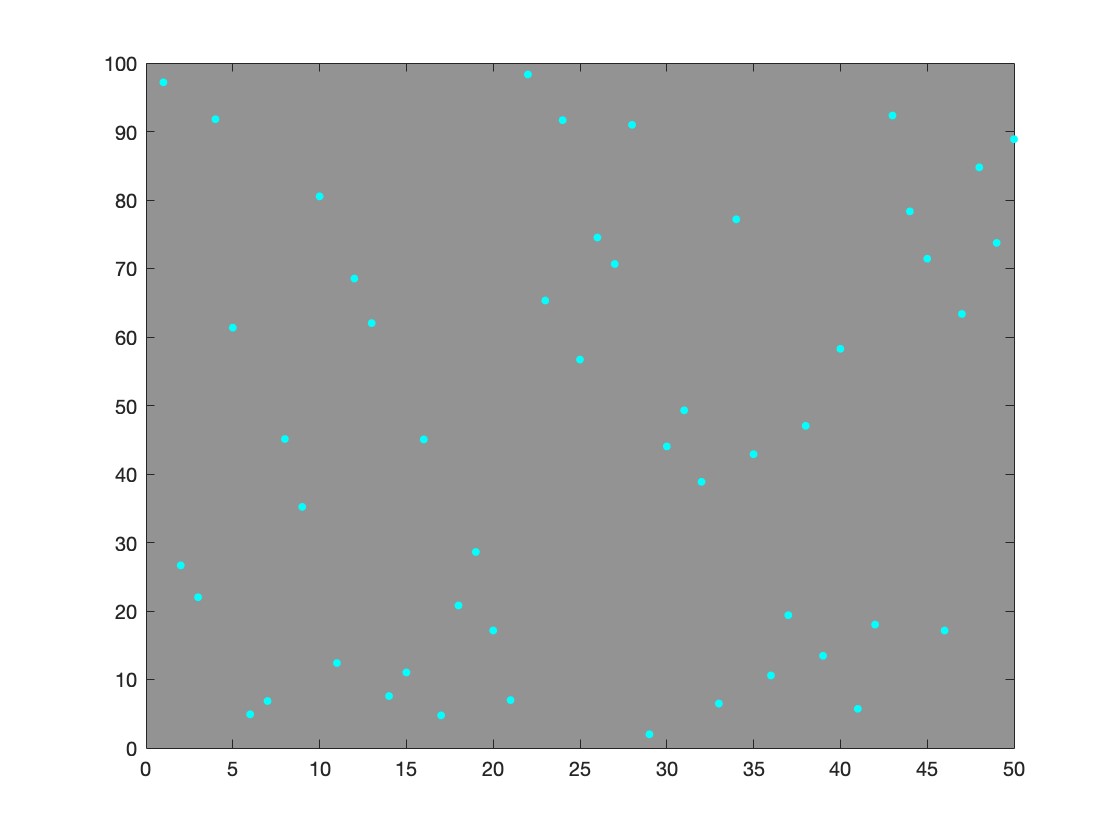


## Figure 8. Raw plots for **Breast Cancer Dataset** Figure 9. Clustering results for **Breast Cancer Dataset**

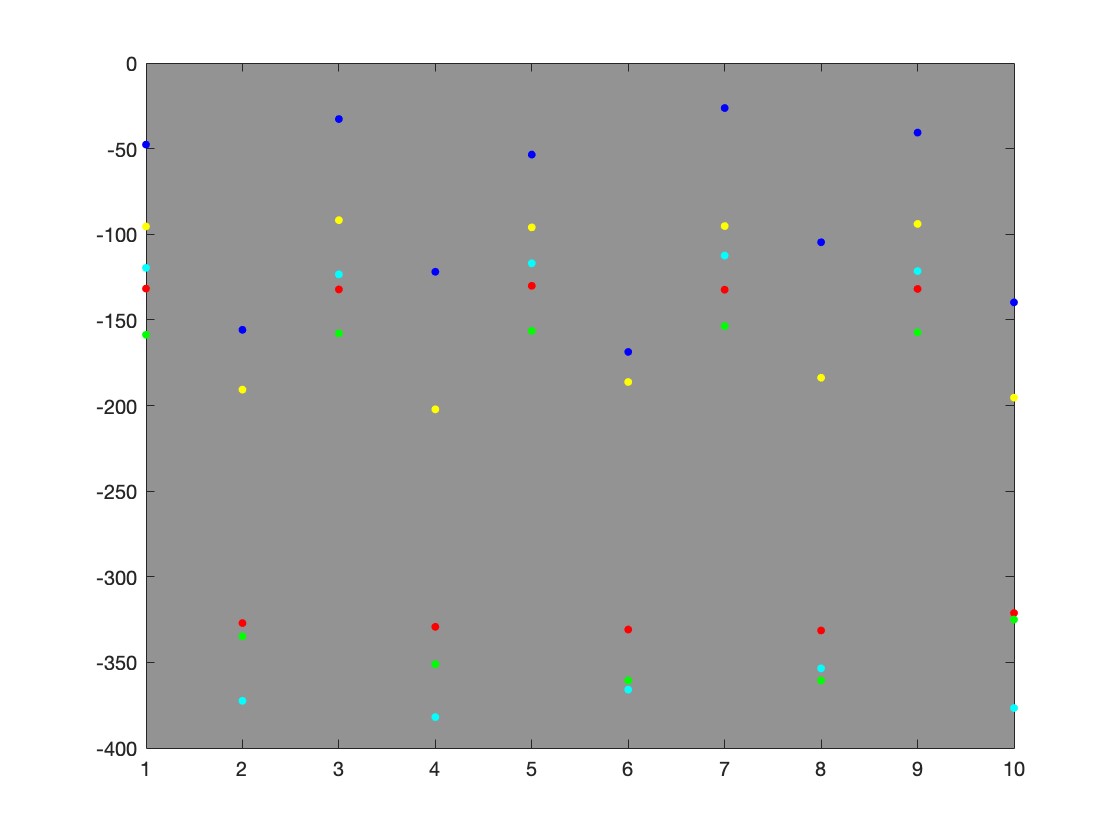
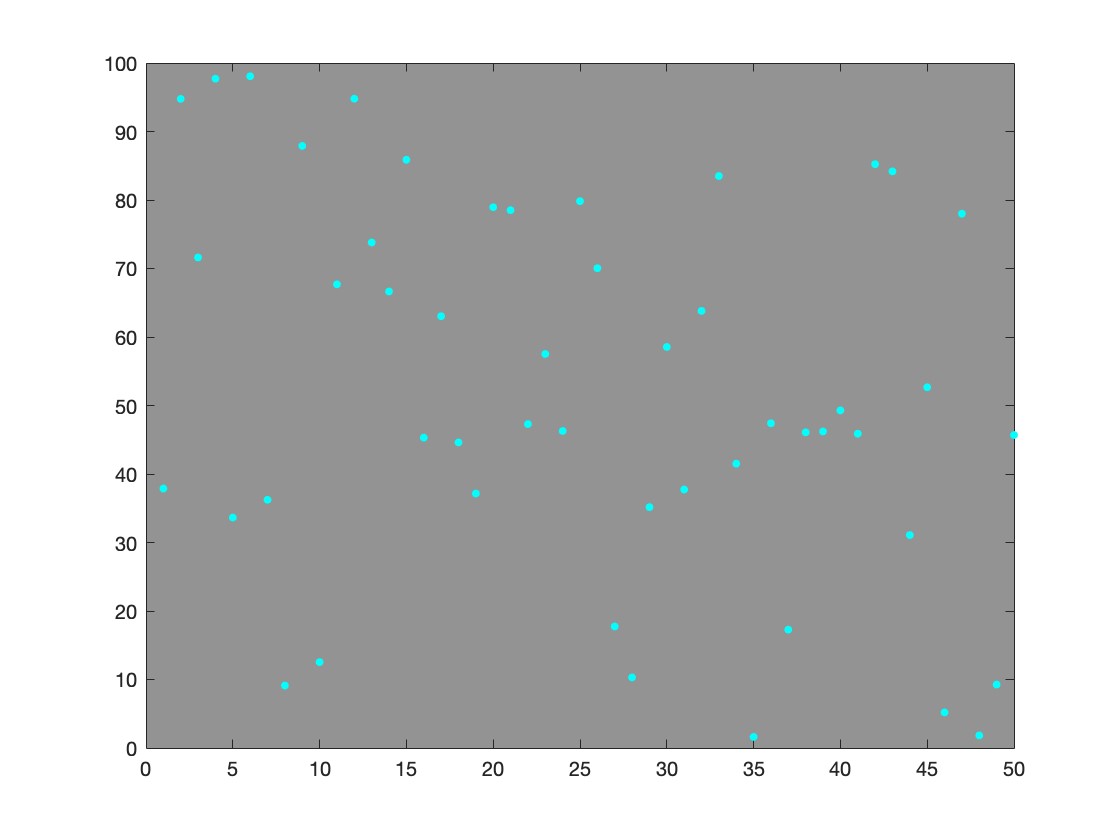


## Figure 10. Raw plots for **Cervical Cancer Dataset** Figure 11. Clustering results for **Cervical Cancer**

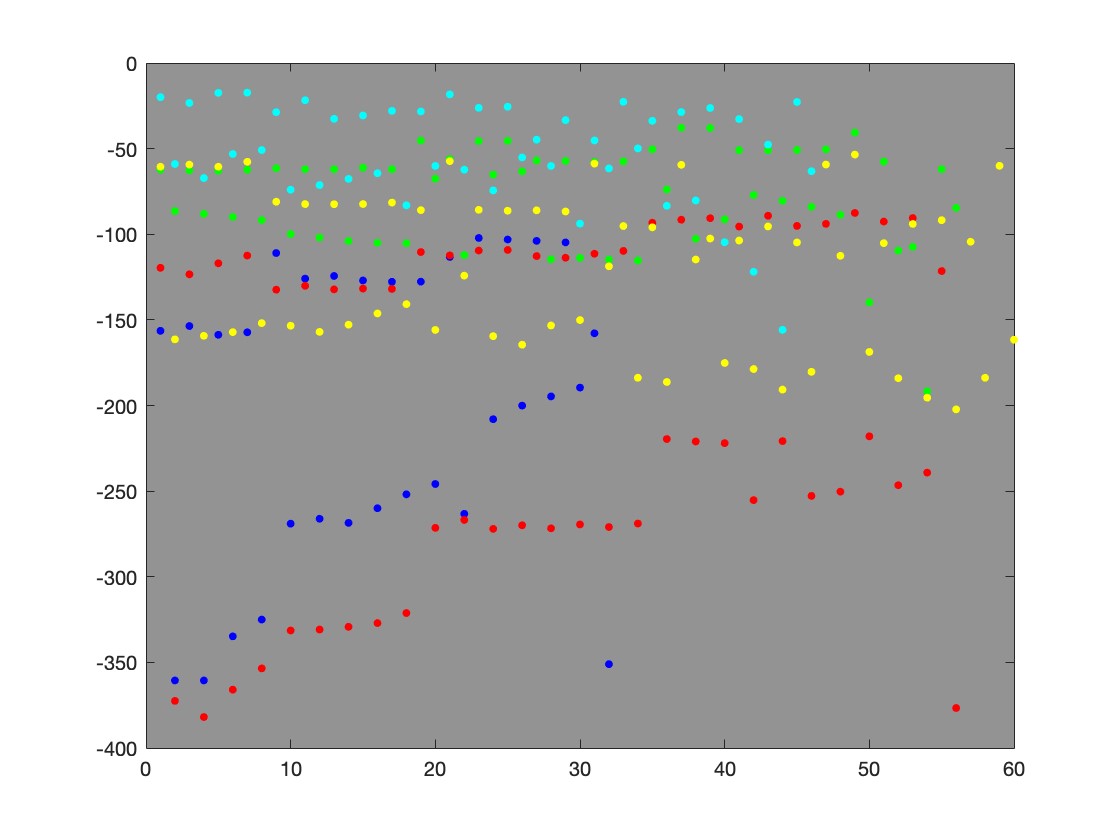
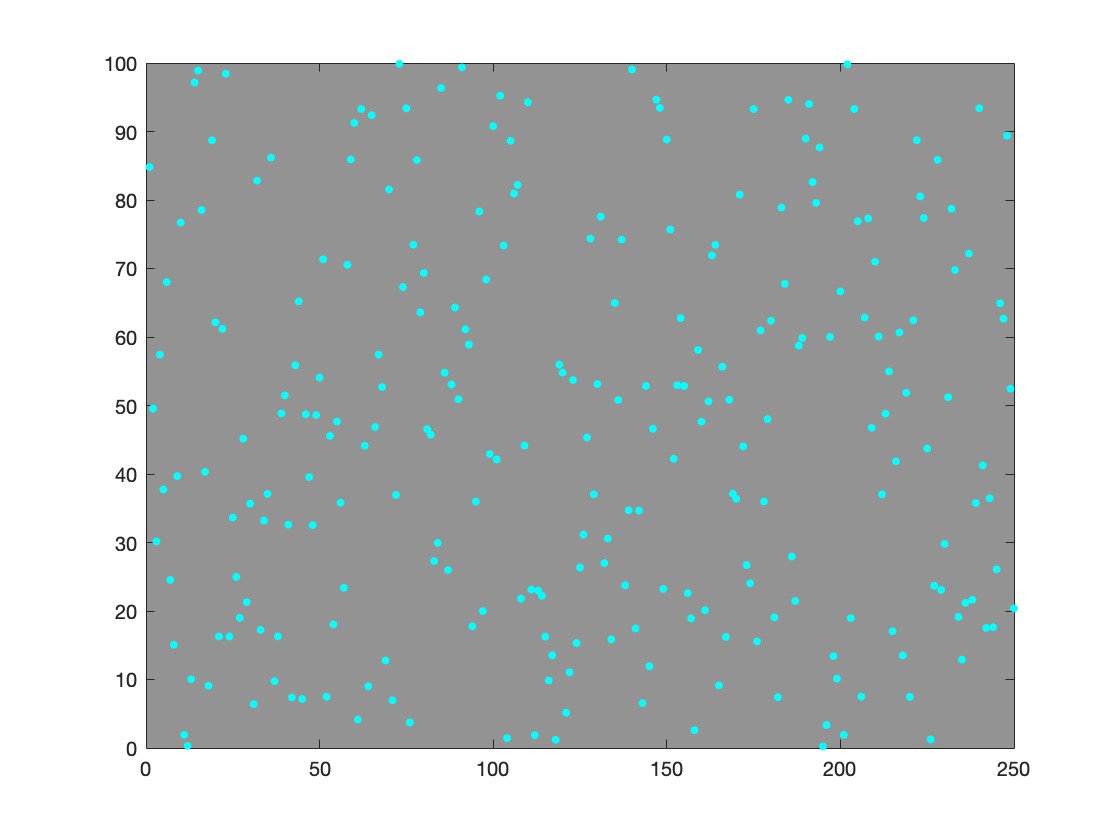
**Dataset**



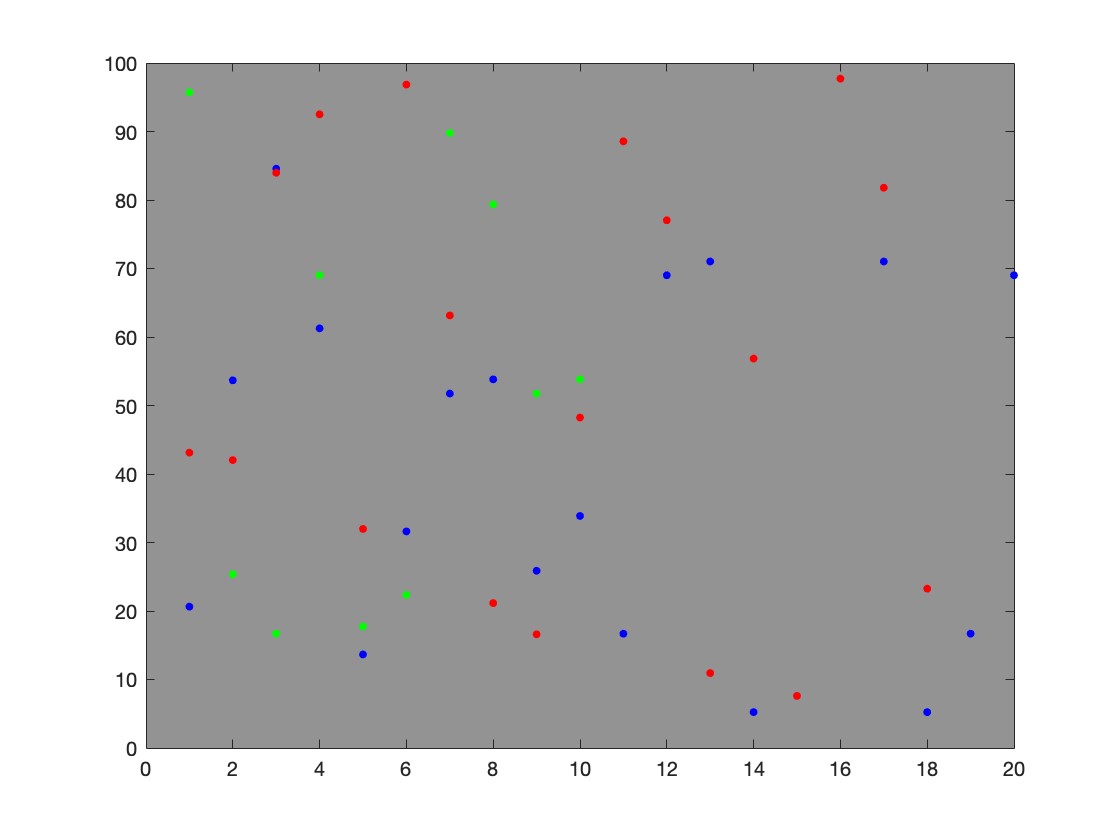
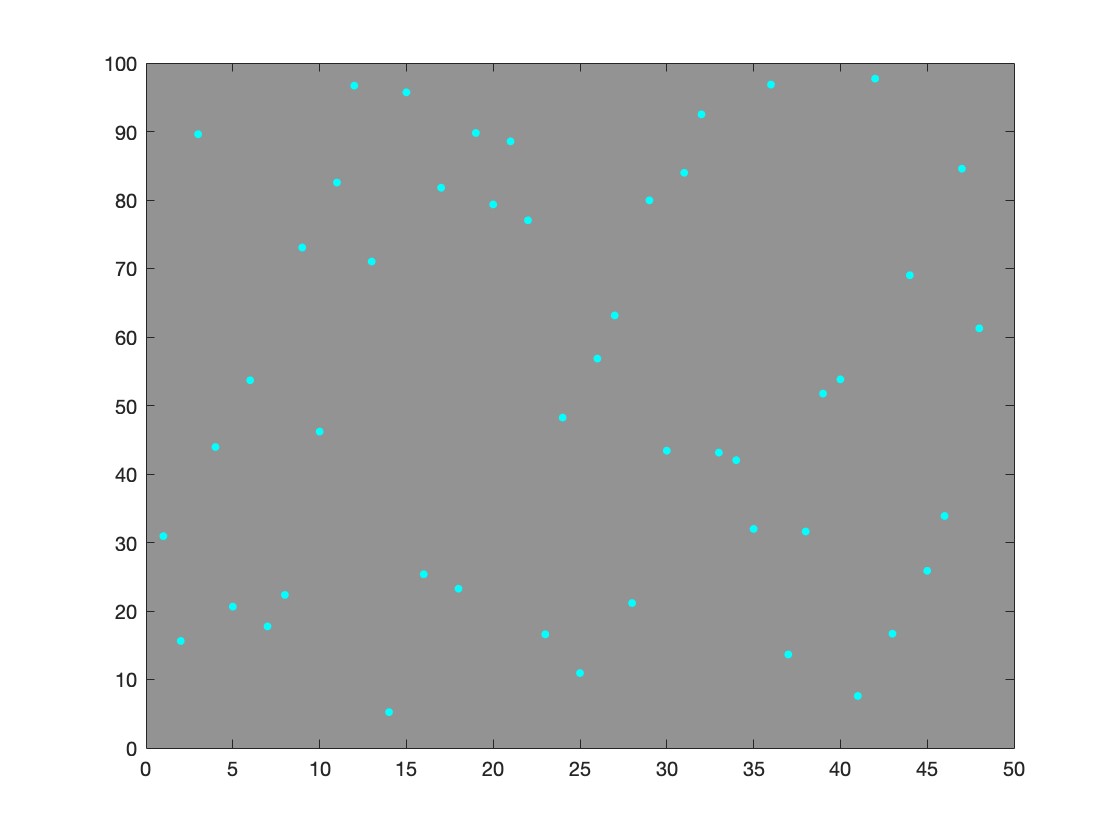
## Figure 12. Raw plots for **QCM3 Dataset** Figure 13. Clustering results for **QCM3 Dataset**



## Figure 14. Raw plots for **QCM6 Dataset** Figure 15. Clustering results for **QCM6 Dataset**



## Figure 16. Raw plots for **QCM Dataset** Figure 17. Clustering results for **QCM Dataset**



## Figure 18. Raw plots for **Documents Topics Dataset** Figure 19. Clustering results for **Documents Topics**

**Dataset**

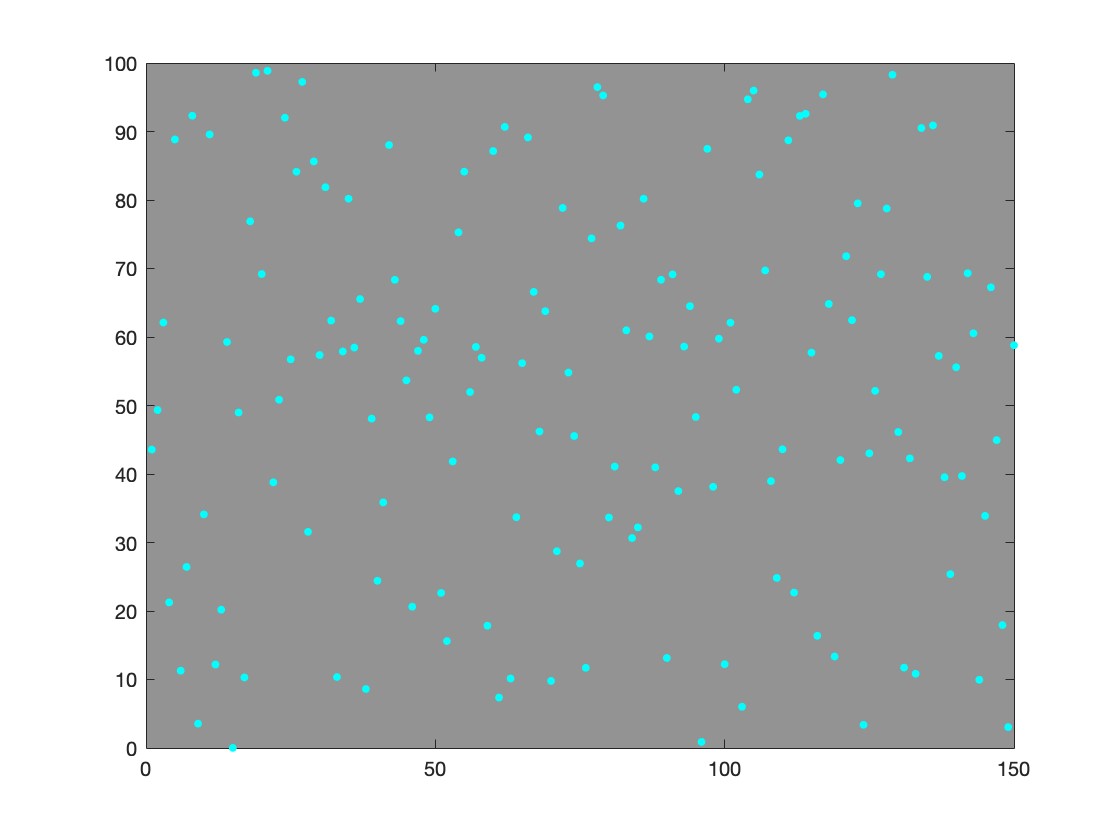


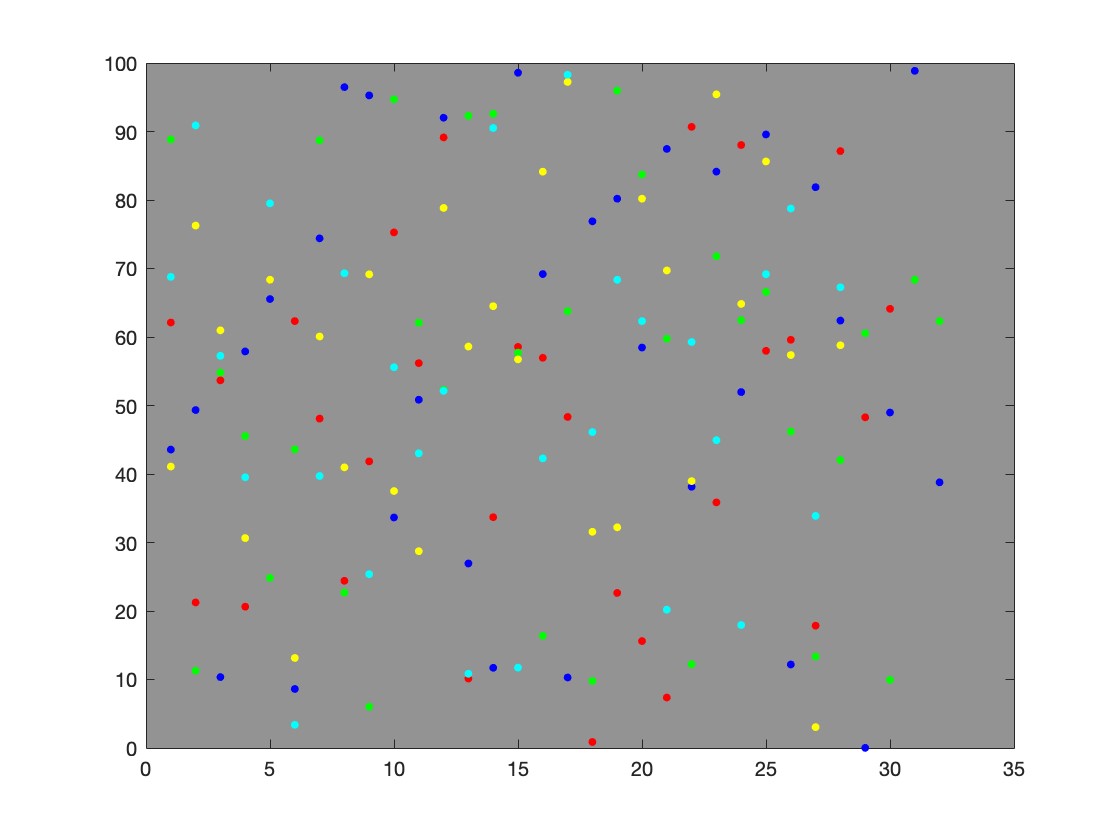
Figure20.Rawplotsfor

**Reuter**

**50**

**50**

**Dataset**



Figure

21.

Clustering

results

for

**Reuter**

**50**

**50**

**DatasetDataset**

From visualizations provided in Section 4.2, one clear observation is that the visualization of cluster is much better with datasets that have larger amount of data, such as *HTRU* in Fig. 7. I deduce the reason is that as described in previous sections, this algorithm enables some data points, the *Dolphin* objects, to move towards other data points (*Prey*). Therefore, the more data points there are, the more data points may move, and therefore the denser the final cluster will be.

Another thing to note from Section 4.2 is that different from *k*−means and some other clustering algorithm, this algorithm’s visualized cluster is more “linear”. This feeling is most obvious in Fig. 5, Fig. 7, Fig. 11, and Fig. 17. Based on the performance measures calculated in Section 4.1, we may deduce that this algorithm is good at clustering linear-shape datasets.

Last thing to note here is that from Fig. 9, we can see that the clusters are overlapping each other. Looking at data points in all, we may find there is no clear cluster pattern. However, based on the performance measures in Section 4.1 on *Breast Cancer Dataset*, we can see that the predicting accuracy is pretty well. From this we may deduce this algorithm is good at clustering dataset that do not have clear cluster pattern at the first glance.

# REFERENCES

[1] Wu, Tq., Yao, M. Yang, Jh. Dolphin swarm algorithm. Frontiers Inf Technol Electronic Eng 17, 717–729 (2016). https://doi.org/10.1631/FITEE.1500287.

W. Qiao and Z. Yang, ”An Improved Dolphin Swarm Algorithm Based on Kernel Fuzzy C-Means in the Application of Solving the Optimal Problems of Large-Scale Function,” in IEEE Access, vol. 8, pp. 2073-2089, 2020, doi: 10.1109/ACCESS.2019.2958456.

A. Bellaachia and A. Bari, ”A flocking based data mining algorithm for detecting outliers in cancer gene expression microarray data,” 2012 International Conference on Information Retrieval Knowledge Management, Kuala Lumpur, 2012, pp. 305-311, doi: 10.1109/InfRKM.2012.6204996.

Bellaachia A., Bari A. (2012) Flock by Leader: A Novel Machine Learning Biologically Inspired Clustering Algorithm. In: Tan Y., Shi Y., Ji Z. (eds) Advances in Swarm Intelligence. ICSI 2012. Lecture Notes in Computer Science, vol 7332. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-31020-1 15.

W. Qiao and Z. Yang, ”Solving Large-Scale Function Optimization Problem by Using a New Metaheuristic Algorithm Based on Quantum Dolphin Swarm Algorithm,” in IEEE Access, vol. 7, pp. 138972-138989, 2019, doi: 10.1109/ACCESS.2019.2942169.

Amic S., Soyjaudah K.M.S., Ramsawock G. (2019) Dolphin Swarm Algorithm for Cryptanalysis. In: Satapathy S., Bhateja V., Somanah R., Yang XS., Senkerik R. (eds) Information Systems Design and Intelligent Applications. Advances in Intelligent Systems and Computing, vol 863. Springer, Singapore. https://doi.org/10.1007/978981-13-3338-5 15.