


ÇUKUROVA UNIVERSITY ENGINEERING FACULTY DEPARTMENT OF COMPUTER ENGINEERING

Subject

A Comparison Between the Firefly Algorithm and Whale Optimization Algorithm

By

Begüm BOLAT

2017555010

Advisor

Havva Esin ÜNAL, Instructor Doctor of Engineering

Department of Computer Engineering, Çukurova University

June 2022

ADANA

Abstract

When large or difficult problems are encountered, computers are often used to find solutions. But when these problems become too large and complicated, traditional methods may not be enough to find the answers. In order to solve these difficult problems, optimization algorithms can be used, which are inspired by nature and offer us the best solution among all solutions under the given conditions. These algorithms try to imitate the living things in nature and their processes and to transform the techniques of solving a problem into an algorithm. In this study, the optimal results obtained by applying the swarm based, metaheuristic Firefly Algorithm and metaheuristic Whale Optimization Algorithms on different datasets, and results of the clustering operation on the non-optimized data were compared.

Table of Contents

1.Introduction

Almost since computer were invented, problems have been adapted and solved with computers. Computers also provide the necessary tools to solve many problems that are very difficult or nearly impossible to solve, faster and more accurately than their calculations would be by hand. However, time is important factor in solving problems. Technically, the problems are solved, but some problems can be solved in a very long time. For example, calculating the result in one second for a problem may take a long time, while for another problem, the result calculated in one day may be considered a short time. How reasonable the time it taken to calculate the best answer is, however, depends on the problem. For example, when we think of an airline company, there are significant economic benefits for the company to be able to fly the most convenient route with minimum empty seats and airport waiting time. Because the demands and needs of thousands of airplanes, airports and passengers change every day, computers cannot solve this problem quickly and precisely enough. For such large-scale problems, it is impossible to find the definitive answer in a suitable time, but we can find approximately the result. This makes use of a class of programs called artificial intelligence that aims to optimize any problem as quickly as possible, although they do not guarantee the best results. Likewise, the time required to locate a chemical leak source is very limited, we can say seconds. Considering all this, if standard approaches cannot meet the time needed, artificial intelligence will be a good option.

Artificial Intelligence has been inspired by natural processes such as ant colonies, bees, fireflies, whales, flowers, black holes, wolves etc. Whale Optimization Algorithm, which was inspired by the feeding of Humpback whales, and Firefly Algorithm, which emerged as a result of examining and modeling firefly swarm, are a kind of artificial intelligence. In this study, these two algorithms are compared by applying them on the same datasets. There are many comparison test problems for these two algorithms. In our study, we got the accuracy scores by running the wine and iris datasets first clustering on the optimized data, then we clustered on the unoptimized version of the same datasets and got the accuracy results again. As a result of both comparisons, it has been observed that the clustering process applied on the optimization algorithms we use provides us with a higher success rate.

2.Metarials and Methods

After I talked to my advisor about optimization algorithms and some projects developed using these algorithms, I have started to search different kinds of optimization algorithms which these algorithms list given by advisor. I have chosen Firefly Algorithm and Whale Optimization Algorithms. Then, Python was selected for programming language because Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms. [1] First, installed Python version 3.9.7 from Web site, www.python.org, and the site has documentation for developers or beginners. Then, I have decided to continue the study on Jupyter Notebook, which is the original web application for creating and sharing computational documents. It offers a simple, streamlined, document-centric experience. [2] To start with downloaded Anaconda that offers the easiest way to perform Python, data science and machine learning on a single machine [3], for Jupyter notebook. Used in the study Anaconda version is 4.11.0. After installation opened Anaconda Navigator and launched Jupyter Notebook. After entering Jupyter Notebook, we create a new kernel file with .ipynb extension with the new button and created the necessary working environment to apply our algorithms. For the next step, two different datasets selected were chosen to be implemented on algorithms. These are İris and Wine datasets. Installed them from Kaggle. İris flower dataset is a multivariate data set introduces by the British statistician a biologist Ronald Fisher in his 1936 paper The use of multiple measurements in taxonomic problems as an example of linear discriminant analysis.[4][5] The dataset consists of 50 samples from each of three species of Iris; Iris setosa, Iris virginica, Iris versicolor. Four features were measured from each sample: the length, and the width of sepals and petals in centimeters.[4]

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Figure 2.1: Iris dataset's first 5 data

Iris dataset shape is (150,5). That means it has 150 data and five columns. As said before, there is three species and each of them contains 50 data. In figure 1.2, we can see the graph of the data classifying according to types with count plot.

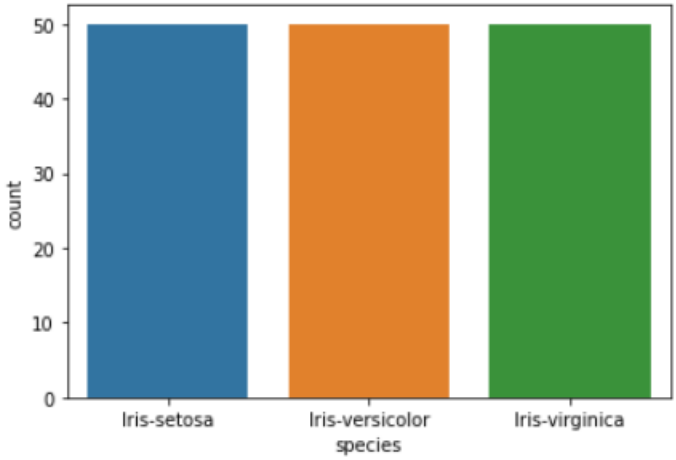


Figure 2.2: Iris data's count by species

In Figure 1.3, excluding the species column, the data contained in the 'sepal_length, sepal_width, petal_lenght, petal_width' columns in the data set can be turned into graphs with the box plot, and it can be displayed in which parts the numerical data is concentrated and mean values.

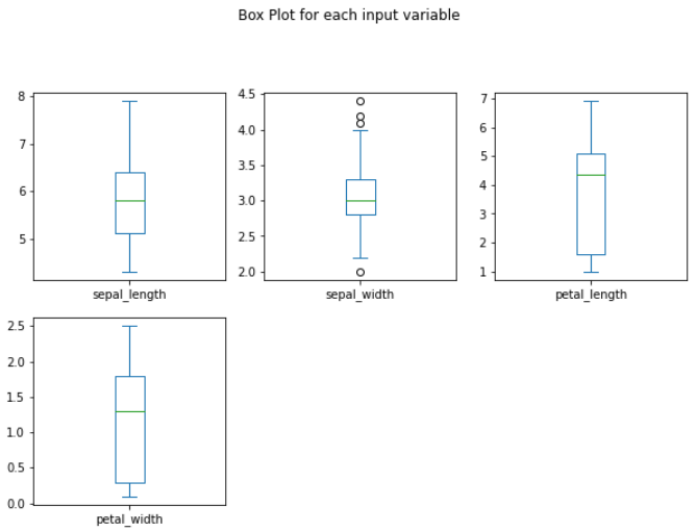


Figure 2.3: Box plot for each column

The other dataset is wine dataset. These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.[6] The wine dataset's shape is (178, 14), these numbers refer to it includes 170 wine samples and 14 columns. This dataset, which has three different classes, is divided into classes 1,2,3 and we can easily see how many wine samples are in each class in the box plot in Figure 1.4. There are 59 wine samples from the 1st class, 71 wine samples from 2nd class and 48 wine samples from the 3rd class.

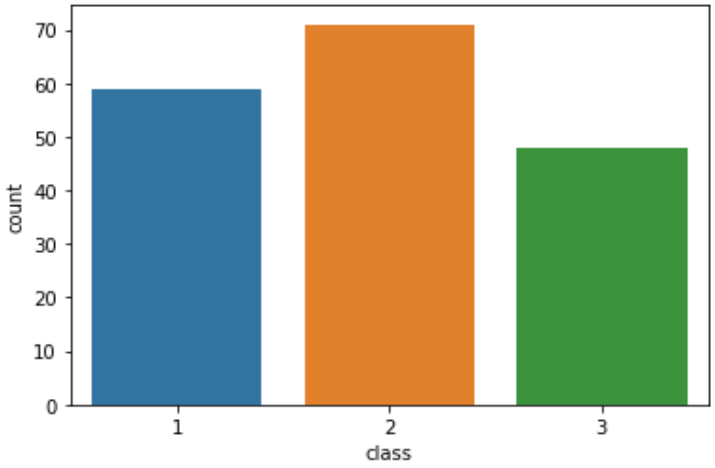


Figure 2.4: Wine samples grouped by their class

In Figure 1.5, these box plots show, dataset's columns 'Alcohol, Malic_acid, Ash, Alcalinity_of_ash, Magnesium, Total_phenols, Flavanoids, Nonflavanoid_phenols, Proanthocyanins, Color_intensity, Hue, OD280/OD315_of_diluted_wines, Proline' and each sample's numerical values for each column and their mean values. These two datasets were run on a laptop with an Inter Core i7-8750H processor with selected Firefly Algorithms and Whale Optimization Algorithms.

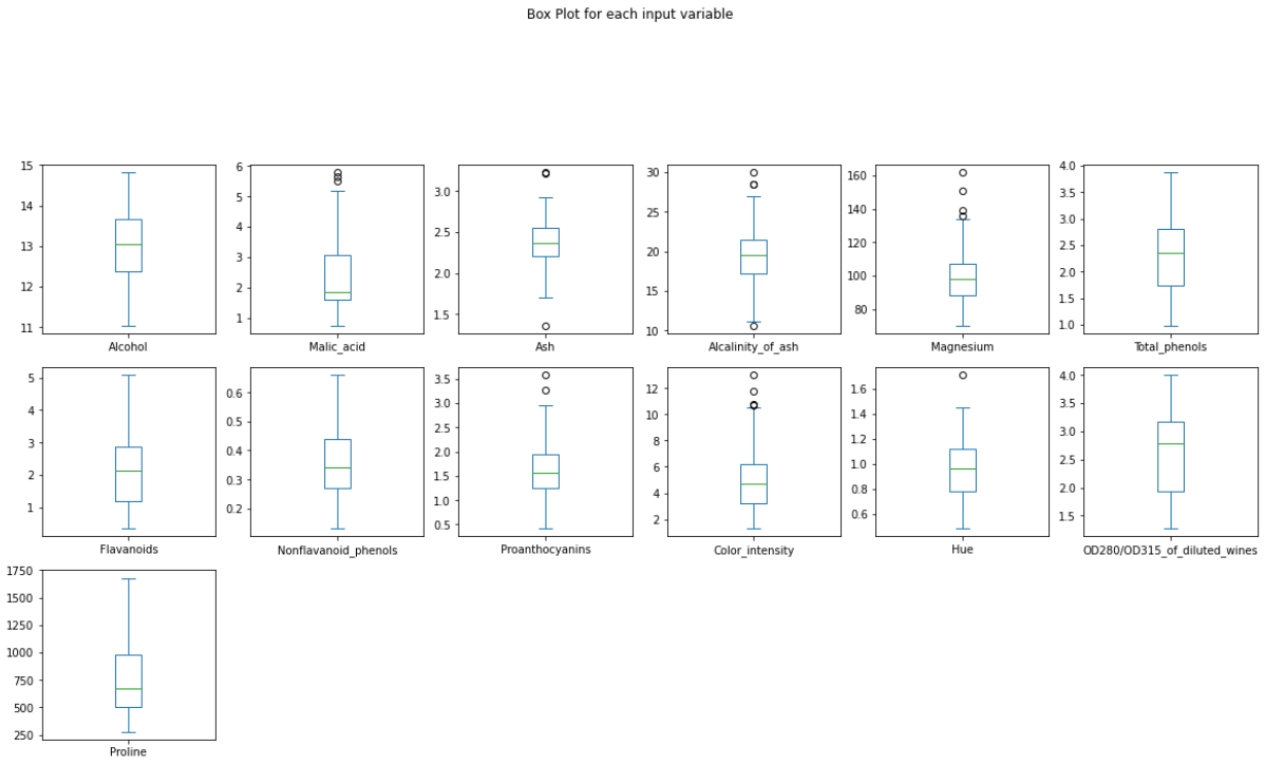


Figure 2.5: Box plot for each column

The clustering problem is a non-well-defined NP-hard problem where the basics idea is to find hidden patterns in our data. There is not a formal definition of what is a cluster, but it's associated with the idea of grouping elements in such a way that we can differentiate elements in distinct groups. There are different families of algorithms that define the clustering problem in distinct ways. A classical way to define the clustering problem, often seen in literature, is to reduce it to a mathematical problem known as finding a k-partition of the original data. Finding a

k-partition of a set S, defined as finding k subsets of S, which obeys two rules: first one is the intersection of any distinct of those subsets is equal to the empty set. Other one is the union of all k subsets is equal to S., at the end of this partitional clustering procedure, we want to find distinct subsets of our original dataset, in such a way that, no instance will belong to more than one group. The output of the clustering procedure is a set of centroids. Centroids are basically the representative entities of each group, so if we want a k-partition of our data, then we will have k centroids. In this study, used the K-Means clustering algorithm, which is an unsupervised learning, for the clustering process.

3.Global Optimization

3.1 - Optimization Algorithms:

Optimization can be defined mathematically as the process of finding values that make an objective function the most optimal (maximum or minimum) under given restrictive conditions depend on the various variables. In a problem, it is to find the optimal variables by choosing the best among the alternatives provided under certain conditions and making the given objective function maximum or minimum. With the design of machines that offer optimum solutions, offer optimal solutions to optimization problems such as businesses or a product can give maximum efficiency and production with minimum costs, which investment tools should people invest their money in order to get maximum profit, sensor placement to create maximum magnetic field, optimizing the electric charge flow, for a robot to follow the minimum path without hitting obstacles etc.

Optimization algorithms, which can be applied in computer engineering, design engineering, robotics, image processing and many other fields, are divided into two sub-branches as deterministic and stochastic. While deterministic algorithms are algorithms that can do contain any operator to create randomness in their structures, stochastic algorithms are generally divided into two as heuristic and metaheuristic algorithms. Since classical method are insufficient in solving optimization problems, new solution methods have been sought and heuristic methods have been started to be studied and many heuristic and metaheuristic algorithms have been developed for the solution of complex and difficult optimization problems, especially in recent

years. Heuristics Algorithms achieve quality results in an acceptable time frame by trial and error even in very difficult optimization problems; but they never guarantee to find the optimum result. The successful way of problem solving (such as finding the shortest way to the food source) of living things in nature is examined by researchers, and optimization techniques are created by trying to imitate their behavior in the herd. These algorithms, inspired by the functioning of various organisms in nature and included in the meta-heuristic category, have opened up new opportunities for solving problems that are difficult to solve and for which exact solution methods are often insufficient. In biological systems, many algorithms based on herd intelligence have been developed by examining and modeling the interaction of the individual with her environment and other individuals and their common behaviors. There is increasing interest in Metaheuristic Optimization Algorithms because of the drawbacks of traditional optimization paradigms, such as being stuck at the local optimum and the need to specify the search space. Some examples of Metaheuristic Optimization Algorithms can be given as prey search which are also discussed in the Whale Optimization Algorithm, ant colony, artificial bee colony, Firefly Algorithm.

3.2 – Firefly Algorithm:

The Firefly Algorithm (FA) is a natural Metaheuristic Optimization Algorithm inspired by the brightness-sensitive social behavior of fireflies in nature. It was presented to the scientific world by Xin-She Yang in 2008.[10] Fireflies are considered unisex and their attraction to each other is directly proportional to their brightness. The less bright firefly is attracted by the brighter one. The greater the distance, the less attractive the attraction. Also, if a firefly can not find a brighter firefly, it will make a random move. A firefly's primary purpose in flashing its lights is to generate a signal to attract the attention of other fireflies.

If we look at the corresponding actions of the firefly's inspired behavior in the algorithm, first the random positions of the fireflies are determined and their brightness is calculated, then the fireflies move towards the brighter fireflies and return to the determined number of repetitions. According to the order of operations corresponding to these behaviors in the algorithm, first random solution sets are created. Each solution set contains as many elements as

the number of parameters. The fitness values of the created solution sets are found, then the improvement formula is applied to all the solution sets in order, and in the last stage, the previous stage is returned until the maximum iteration is reached.

3.2.1 – The Algorithm:

There are several definitions in the content of the algorithm:

1. Problem: It is the problem we are looking for a solution to in the algorithm and is usually determined by a mathematical formula.
2. Fitness Function: The formula for finding the goodness level of the solution sets we found for the problem in the algorithm. ($X_1^2 + X_2^2$) Sphere Function.
3. Population Size: It is the sum of the number of fireflies in the population. That is, the number of solution sets in one iteration. (10)
4. Parameter Size: It refers to the number of unknown variables in our fitness function. (X_1, X_2)
5. Parameter Range: These are the largest and smallest values that parameters can take. [-50,50]
6. Maximum Iteration: Specifies the maximum number of cycles the algorithm will run. (100)
7. α (alpha): Random variable. It can be taken between 0 and 1. (α=0.5)
8. γ (gamma): The constant absorption coefficient can be taken between 0.01 and 100. (γ = 1.0)

First, random locations are assigned to the fireflies. For each solution set, a random solution set is created according to the smallest and largest parameter values as well as the parameter size. The fitness value is calculated with the fitness function. **X_{ik} = lb + rand (0,1) (ub – lb)**, i is index number of selected solution set, k is the index number of the selected in the solution set where i is selected, X_{ik} is current parameter number of the selected solution set, lb means lower bound, which is minimum number the parameter takes and ub means upper bound, which maximum number the parameter can take. Each firefly controls all other fireflies, and while it moves towards them depending on the formula for self-bright fireflies, it performs a random movement if it is not bright. Since the interactions of each firefly with another firefly will be discussed, research with a Big(O) value of n^2 (i.e., nested loop) is carried out. **$I = 1/r^2$** is inverse square law. I is intensity and r is distance. Since the intensity of light decreases with distance, the brightness of fireflies is felt less by distant fireflies. The light intensity (I) is the intensity of the light source at a certain distance. I can be determined in Equation 3.2.1 as follows [12]:

$$I = I_0 e^{-\gamma r}$$

Equation 3.2.1

where r is the distance between fireflies and I_0 is the light intensity at distance (r = 0). The attractiveness of fireflies is defined by Equation 3.2.2 as follows [12]:

$$\beta = \beta_0 e^{-\gamma r^2}$$

Equation 3.2.2

where β_0 is the attractiveness of a firefly at distance (r=0). If a firefly (i) has less brightness, the movement towards the brighter firefly (j) can be determined by Equation 3.2.3 as follows [12]:

$$X_i = X_i + \beta_0 e^{-\gamma r_{ij}^2} (X_j) - \beta_0 e^{-\gamma r_{ij}^2} (X_i) + \alpha * \varepsilon_i$$

Equation 3.2.3

where X_i is a less bright firefly and X_j is a brighter one. That the entire algorithm has been covered, a summary in pseudocode is given in Figure 3.2.1.

Firefly Algorithm

Given:
Objective function f(**x**), where **x** = (x_1, x_2, ..., x_d) for d dimensions.
Light intensity I_i at $\mathbf{x}_i$ is determined by f($\mathbf{x}_i$)
Define α, β, γ, δ

Give random locations for N fireflies in D dimensions
Initial evaluation of all N fireflies
while (End condition not met)
    Increment t
    for i = 1 **to** NumFireflies
        for j = 1 **to** NumFireflies
            if ($I_j > I_i$)
                Move firefly i towards j in d-dimension
            end if
        end for j
        Evaluate new solution and update light intensity
        Check if the best position found so far
    end for i
    Reduce alpha
    Sort the fireflies
end while
Find the best firefly

Figure 3.2.1: FA Pseudocode [8]

3.2.2 – The FA Code:

The first FA application is over the iris dataset. In Figure 3.2.2, import pandas library for reading csv file and then adding csv file with read_csv function.

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
iris = pd.read_csv("iris.csv")
```

Figure 3.2.2

In Figure 3.2.3, except for the species column of the iris dataset, we assign the other columns to the X variable and assign only the species column to the variable y. Then using the Standard Scaler, the data we assigned to the X variables are brought to the normal distribution range with a mean of 0 and a variance of 1. After that, split data using train_test_split, half train and half test.

```
X = iris.drop('species',axis = 1)
y = iris.species
```

```
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X)
X = scaler.transform(X)
```

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test= train_test_split(X, y, train_size = 0.5, random_state = 0)
```

Figure 3.2.3

In Figure 3.2.4, we send the separated data, upper and lower bound values, size, number of fireflies and max generation numbers to the function we defined as FFA.

```
IterationNumber,BestQuality,accuracy = FFA(X_train,X_test,y_train,y_test,-50,50,4,50,10)
```

Figure 3.2.4

Inside FFA function, shown in Figure 3.2.5, we defined general parameters which are number of fireflies, dimension, lower and upper bounds, max generation, alpha, beta and gamma values.

```
def FFA(X_train,X_test,y_train,y_test, lb, ub, dim, n, MaxGeneration):

    # General parameters

    # n=50 #number of fireflies
    # dim=30 #dim
    # lb=-50
    # ub=50
    # MaxGeneration=10

    # FFA parameters
    alpha = 0.5  # Randomness 0--1 (highly random)
    betamin = 0.20  # minimum value of beta
    gamma = 1  # Absorption coefficient
    if not isinstance(lb, list):
        lb = [lb] * dim
    if not isinstance(ub, list):
        ub = [ub] * dim

    zn = numpy.ones(n)
    zn.fill(float("inf"))
```

Figure 3.2.5

Then we randomly place the fireflies with for loop within the dimension number we defined. The code shown in Figure 3.2.6

```
# Determining random locations for fireflies
ns = numpy.zeros((n, dim))
for i in range(dim):
    ns[:, i] = numpy.random.uniform(0, 1, n) * (ub[i] - lb[i]) + lb[i]
Lightn = numpy.ones(n)
Lightn.fill(float("inf"))
```

Figure 3.2.6

Also in Figure 3.2.7, **alpha_new** function returning new alpha value for using in main loop.

```
def alpha_new(alpha, NGen):
    #calculate a new value of alpha
    delta = 1 - (10 ** (-4) / 0.9) ** (1 / NGen)
    alpha = (1 - delta) * alpha
    return alpha
```

Figure 3.2.7

Then, starting out main loop with the while loop that will start from 0 and continue until the number of iterations. In this main loop, we basically calculate the fireflies tending to more fireflies tending to more fireflies, the attraction of fireflies, the movement of fireflies towards a brighter firefly, or the random movements of fireflies. As can be seen in Figure 3.2.8, in the main part, which starts with the while loop, we apply KMeans operation according to the train ones from our data that we send by splitting it into the FFA function. After that, we can see the performance of our model by dividing the new data we clustered with KMeans into 10 different subsets using KFold, using cross_val_score, and averaging the Accuracy and MAE values we obtained as a result of our cross-validation process.

```
# Main loop
t = 0
while t < MaxGeneration:
    for k in range(0, MaxGeneration):  #start iterations

        #This line of reducing alpha is optional
        alpha = alpha_new(alpha, MaxGeneration)

        kmeans = KMeans(n_clusters = i).fit(X_train,y_train)
        cv = KFold(n_splits = 10, shuffle = True, random_state = 15)
        cv_accuracies = cross_val_score(kmeans,  X_test, y_test, cv = cv,scoring = 'accuracy')


        accuracies = cv_accuracies.mean()
        fitness_value = (1 - accuracies) * 100
```

Figure 3.2.8

After clustering our data with KMeans and performing the fit operation, we predict new solutions for the 50 fireflies we have defined and assign the predicted values to the **Ligthn** variable. Shown in Figure 3.2.11. Then we rank the fireflies in Ligthn according to their light intensity. We separate the first element as the best values from the values we ranked. After applying the distance formula that we defined to place all other fireflies in better positions, we apply the formula in Equation 3.2.3 which calculates the movement to brighter one. After all these stages, the FFA function gives us the accuracy value of 1.0 and returned best fitness value is 16 at 10^{th} iteration for the **iris** dataset in Figure 3.2.9. For **wine** dataset the FFA function gives us the accuracy value 1.0 and returned the best fitness value is 24.0 at 10^{th} iteration in Figure 3.2.10.

```
IterationNumber,BestQuality,accuracy = FFA(X_train,X_test,y_train,y_test,-50,50,4,50,10)
```

```
[0.65, 0.71, 0.72, 0.84, 0.87, 0.87, 0.88, 0.97, 0.98, 1.0]
At iteration is: 10 the best fitness is 16.0
accuracy: 1.0
```

Figure 3.2.9

```
IterationNumber,BestQuality,accuracy = FFA(X_train,X_test,y_train,y_test,-50,50,13,50,10)
```

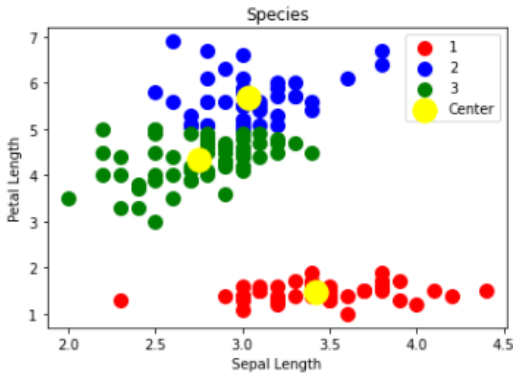
```
[0.61, 0.72, 0.75, 0.76, 0.8, 0.93, 0.94, 0.96, 0.99, 1.0]
At iteration is: 10 the best fitness is 24.0
accuracy: 1.0
```

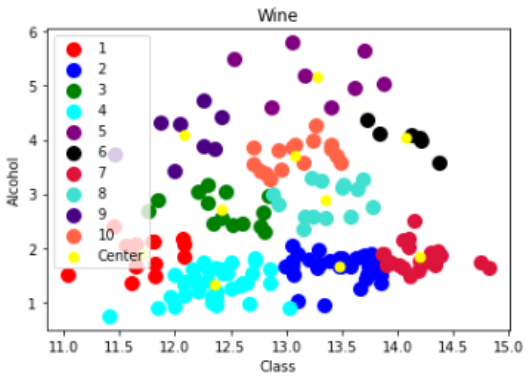
Figure 3.2.10

```
        #Evaluate new solutions (for all n fireflies)
        for i in range(0, n):
            zn[i] = kmeans.predict(ns[i, :].reshape(1,-1))
            Lightn[i] = zn[i]

        #Ranking fireflies by their light intensity/objectives
        Lightn = numpy.sort(zn)
        Index = numpy.argsort(zn)
        ns = ns[Index, :]

        #Find the current best
        nso = ns
        Lighto = Lightn
        nbest = ns[0, :]
        Lightbest = Lightn[0]

        #For output only
        fbest = Lightbest
        convergence.append(fbest)
        array.append((100-fbest)/100)

        #Move all fireflies to the better locations
        #  [ns]=ffa_move(n,d,ns,Lightn,nso,Lighto,nbest,...
        #    Lightbest,alpha,betamin,gamma,Lb,Ub);
        scale = []
        for b in range(dim):
            scale.append(abs(ub[b] - lb[b]))
        scale = numpy.array(scale)
        for i in range(0, n):
            #The attractiveness parameter beta=exp(-gamma*r)
            for j in range(0, n):
                r = numpy.sqrt(numpy.sum((ns[i, :] - ns[j, :]) ** 2)) #Distance formula
                #r=1
                #Update moves
                if Lightn[i] > Lighto[j]:  #Brighter and more attractive
                    beta0 = 1
                    beta = (beta0 - betamin) * math.exp(-gamma * r ** 2) + betamin
                    tmpf = alpha * (numpy.random.rand(dim) - 0.5) * scale
                    ns[i, :] = ns[i, :] * (1 - beta) + nso[j, :] * beta + tmpf

    t = t + 1
    iterations.append(t)
    accuracy.append((100-fbest)/100)


    IterationNumber = k
    BestQuality = fbest
```

Figure 3.2.11

After finding the accuracy results of the optimized values, we draw scatter graphs using KMeans with n cluster values we determined according to the WCSS graph, the data we split as train test.





In the Figure 3.2.12 and Figure 3.2.13 includes iris and wine datasets, cross validation results.

```
cv = KFold(n_splits = 10, shuffle = True, random_state = 30)
accuracies = cross_val_score(kmeans, X , y , cv = cv)
accuracy_mean = accuracies.mean()
print(accuracy_mean)
accuracy_std = accuracies.std() * 100
print(accuracy_std)
```

```
-1.4975137474422917
39.46074740617141
```

Figure 3.2.12: Iris dataset accuracies

```
cv = KFold(n_splits = 10, shuffle = True, random_state = 30)
accuracies = cross_val_score(kmeans, X , y , cv = cv)
accuracy_mean = accuracies.mean()
print(accuracy_mean)
accuracy_std = accuracies.std() * 100
print(accuracy_std)
```

```
-0.8726428098859917
33.98184731942352
```

Figure 3.2.13: Wine dataset accuracies

To sum up, the clustering process we applied on the optimized data gave us a higher success rate than the one we applied directly on both of iris and wine dataset.

3.3 – Whale Optimization Algorithm:

Whale Optimization Algorithm (WOA), one of the new meta-heuristic optimization algorithms, is inspired by the bubble fishing strategy that humpback whales use while hunting [13]. This algorithm, which is swarm-based such as Firefly Algorithm, Ant Colony Algorithm, has many unique features. Humpback whales, that feed on small shoals of fish, have their own unique hunting strategies. Whales, which from bubble clouds by breathing under the water, gather their prey together in this way [14]. The Figure 3.3.1 below shows a representation of whale's bubble strategy hunting and a real picture of humpback whale's bubble strategy hunting. Hunting strategy in Whale Optimization Algorithm; It's modeled in 3 parts as wrapping around the prey, moving towards the prey and searching for prey.

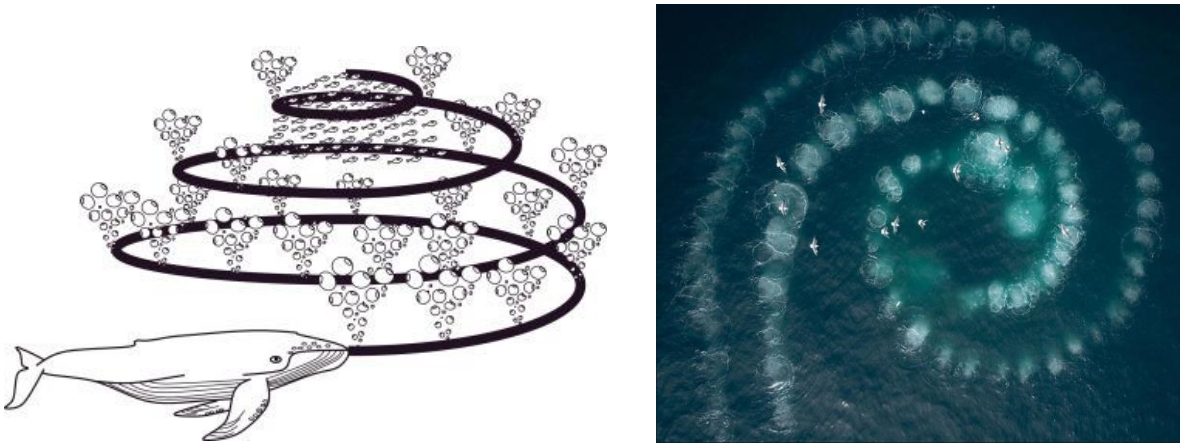


Figure 3.3.1: Humpback Whale Hunting Strategies

3.3.1 – The Algorithm:

First step is wrapping around the prey. In Whale Optimization Algorithm, prey is considered as the optimum solution to be reached. Since the optimum solution is not known in optimization problems, the optimum solution is tried to be obtained by using local or global search spaces. The search continues until certain criteria are met. The optimum solution is considered the best solution reached or a point around it. After the best solution is determined,

the position of the other solutions is updated using the best solution. The mathematical model of prey encircling behavior is shown in Equation 3.3.1 and Equation 3.3.2 [13].

$$\vec{D} = \left| \vec{C}.\vec{X}^{*}(t) - \vec{X}(t) \right|$$

Equation 3.3.1

$$\vec{X}(t+1) = \left| \vec{X}^{*}(t) - \vec{A}.\vec{D} \right|$$

Equation 3.3.2

t, represents the current iteration coefficient vectors A and C, and X^{*} represents the best solution vector.

$$\vec{A} = 2\vec{a}.\vec{r} - \vec{a}$$

Equation 3.3.3

$$\vec{C} = 2.\vec{r}$$

Equation 3.3.4

r, is a random vector and a is a decreasing vector from 2 to 0 over the iterations. Second step is moving towards the prey, whales that find the prey approach their prey in two different ways, such as narrowing the circle around the prey and spiral motion to move towards their prey. These movements are modeled as follows. It was thought that narrowing the circle around the prey would be possible by decreasing the value of **a** in Equation 3.3.3. In Figure 3.3.2 shows the spiral motion of the whales and the position of the best solution. Equation 3.3.5 was created by calculating the distance between the target location and the solution candidate for this motion.

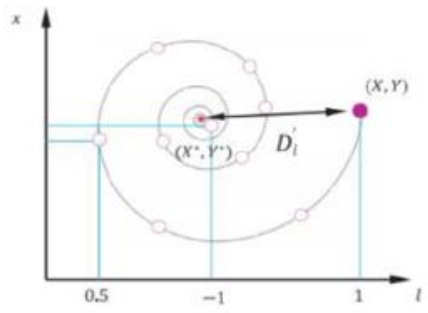


Figure 3.3.2: Spiral Movement

$$\vec{X}(t+1) = \vec{D}'.e^{bl}.\cos(2\pi l) + \vec{X}^*(t)$$

Equation 3.3.5

$$\vec{D}' = \vec{X}^*(t) - \vec{X}(t)$$

Equation 3.3.6

b, is the algorithmic spiral constant, and 1 represents a random number in the range [-1,1]. In the algorithm, which of the spiral motion and linear motion will be made is determined with $^1/_2$ probability as shown in Equation 3.3.7.

$$\vec{X}(t+1) = \begin{cases} \left| \vec{X}^*(t) - \vec{A}.\vec{D} \right|, & p < 0.5 \\ \vec{D}.e^{bl}.\cos(2\pi l) + \vec{X}^*(t), & p \geq 0.5 \end{cases}$$

Equation 3.3.7

p, is a random number in the range [0,1]. Third and the last step is, searching for prey. For the global solution, the new positions of the solution candidates are determined around a randomly chosen solution candidate instead of the known best point. The mathematical model is shown in Equation 3.3.8 an Equation 3.3.9.

$$\vec{D}' = \vec{C}.\vec{X}_{rand} - \vec{X}$$

Equation 3.3.8

$$\vec{X}(t+1) = \vec{X}_{rand} - \vec{A}.\vec{D}$$

Equation 3.3.9

X_{rand} , represents a randomly chosen solution vector. Whether global or local searches will be made is decided according to the value of vector **A**. For vector A, when A>1 or A<1 the point further away from the best point can be selected, these cases are considered as global search and Equation 3.3.8 and Equation 3.3.9 are applied.

Randomly initialize the whale population.
Evaluate the fitness values of whales and find out the best search agent X^*.
while $t < t_{max}$
  Calculate the value of a according to Equation (13)
  for each search agent
    if $h < 0.5$ then
      if $|A| < 1$ then $X(t+1) = X^*(t) - A \cdot D$
      else if $|A| \geq 1$ then $X(t+1) = X_{rand}(t) - A \cdot D''$
      end if
    else if $h \geq 0.5$ then
      $X(t+1) = D' \cdot e^{bl} \cdot \cos(2\pi l) + X^*(t)$
    end if
  end for
  Evaluate the fitness of $X(t+1)$ and update X^*
end while

Figure 3.3.3: Pseudocode of Whale Optimization Algorithm

3.3.2 -The WOA Code:

For Whale Optimization Algorithm, in the first stage, reading csv file with pandas library in Figure 3.3.3, then create X, y variables by separating the class column from the other columns.

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
wine = pd.read_csv("wine.csv")
```

Figure 3.3.3

```
feature_names = ['Alcohol', 'Malic_acid', 'Ash', 'Alcalinity_of_ash','Magnesium',
                 'Total_phenols','Flavanoids','Nonflavanoid_phenols','Proanthocyanins',
                  'Color_intensity','Hue','OD280/OD315_of_diluted_wines','Proline']
X = wine[feature_names]
y = wine['class']
```

Figure 3.3.4

Then, by using MinMaxScaler, we ensure that our data takes values between 0 and 1. So, we subtract the minimum values in that data range from our value and divide by the range values. After that, we send necessary values which are dataset, lower and upper bound value, number of

whales and number of iterations, for the Whale Optimization Algorithm to WOA function. Inside the WOA function, the first step is to randomly position the whales shows in Figure 3.3.5.

```
def WOA(wine, lb, ub, dim ,SearchAgent, maxIter):

    whalePop = []
    for i in range(SearchAgent):
        tmp = np.random.rand(3*dim)*(ub-lb)-(ub-lb)/2
        whalePop.append(tmp)
```

Figure 3.3.5: Random positioning of whales

Additional, defining the parameters required for WOA, we started the process that will continue as many as the number of iterations. Then evaluate the fitness values of whales and find out the best search agent and choose best search agent as **idXFit.**

```
while t<maxIter:

    fitnessPop = []

    for whale in whalePop:
        fitnessWhale = 0
        for p in range(P):
            distances = [norm(iris.iloc[p]-whale[0:13]),norm(iris.iloc[p]-whale[13:26]), norm(iris.iloc[p]-whale[26:39])]
            idxDist = np.argmin(distances)
            fitnessWhale += distances[idxDist]
        fitnessPop.append(fitnessWhale)
        count += 1


    idxFit = np.argmin(fitnessPop)
    bestWhale = whalePop[idxFit]
    fitness_vec.append(fitnessPop[idxFit])

    print("Iteration "+str(t)+", Fitness of best whale "+str(fitnessPop[idxFit]))
```

Figure 3.3.6

In Figure 3.3.7, inside the for loop we update a, A, C, p, l and r values for each search agent. If p variable's value smaller than 0.5, and A's value equal or bigger than 1, we select random agent and update position, or A's value is smaller than 1, we update position of agent. Besides that, if p's value bigger or equal than 0.5, we update position of agent. After these steps, we calculate best fitness for each search agent and update optimal solution. Finally return best search agent and its fitness value.

```
    for i,whale in enumerate(whalePop, 0):

        if i != idxFit:

            a = 2-(2*t)/maxIter
            r = np.ones(3*dim)*np.random.uniform(0,1,1)[0]
            A = 2*a*r-a
            C = 2*r
            p = np.random.uniform(0,1,1)[0]
            l = np.random.uniform(-1,1,1)[0]

            if p<0.5:
                if norm(A)<1:
                    D = np.multiply(C,bestWhale)-whale
                    whalePop[i] = bestWhale-np.multiply(A,D)

                elif norm(A) >= 1:
                    #Phase of exploration
                    XRand = np.random.rand(3*dim) #must not necessarily be in [0,1]
                    #D = np.abs(np.multiply(C,bestWhale)-whale)
                    D = np.multiply(C,bestWhale)-whale
                    whalePop[i] = XRand-np.multiply(A,D)

            elif p>= 0.5:
                #Phase of spiral
                D_ = bestWhale-whale
                whalePop[i] = D_*np.exp(b*l)*np.cos(2*np.pi*l)+bestWhale

            #Bring back whale which is out of search space
            for j,coordinate in enumerate(bestWhale,0):
                if coordinate > 1:
                    bestWhale[j] = 0.5
                elif coordinate < 0:
                    bestWhale[j] = 0.5

    t +=1
    z1,z2,z3 = bestWhale[0:13],bestWhale[13:26],bestWhale[26:39]

return z1,z2,z3
```

Figure 3.3.7

When we run the WOA function, it returns the fitness value for each iteration value, as seen in Figure 3.3.8.

```
dim = len(wine.columns)
z1,z2,z3 = WOA(wine, 0, 1, dim, 10, 100)
```

```
Iteration 0, Fitness of best whale 232.57451372229255
Iteration 1, Fitness of best whale 141.58232328300812
Iteration 2, Fitness of best whale 141.45422540999763
Iteration 3, Fitness of best whale 141.44940029286286
Iteration 4, Fitness of best whale 141.44940029286286
Iteration 5, Fitness of best whale 135.85625979252904
Iteration 6, Fitness of best whale 135.9152363655915
Iteration 7, Fitness of best whale 135.9152363655915
Iteration 8, Fitness of best whale 135.83130874432234
Iteration 9, Fitness of best whale 135.05915117487172
Iteration 10, Fitness of best whale 135.05915117487172
```

Figure 3.3.8

After that part, we describe new array that name is cluster. The column named Cluster contains the index values of the column named cluster by subtracting the positions where the bestWhale value and the fitness value returned to us from the elements in the p^{th} row of wine dataset. Lastly,

obtained the accuracy result by comparing this column with the y column containing the classes. All mentioned parts are shown in Figure 3.3.9 and 3.3.10. Their accuracy score for iris dataset is 0.32, for wine dataset is 0.2.

```
cluster = []
P = len(wine)
for p in range(P):
    distances = [norm(wine.iloc[p]-z1),norm(wine.iloc[p]-z2),norm(wine.iloc[p]-z3)]
    cluster.append((np.argmin(distances)+1))
wine["cluster"] = cluster
```

```
z1
```

```
array([0.55347158, 0.46606714, 0.55521337, 0.52951799, 0.33311273,
       0.24590759, 0.29950317, 0.39241253, 0.18428373, 0.40121586,
       0.28193611, 0.21439785, 0.4204812 ])
```

```
counter = 0
P = len(wine)
for p in range(P):
    if wine.iloc[p].cluster == y.iloc[p]:
        counter += 1
acc = counter / P
print("acc: " + str(acc))
```

```
acc: 0.3258426966292135
```

Figure 3.3.9: Wine dataset's accuracy score

```
cluster = []
P = len(iris)
for p in range(P):
    distances = [norm(iris.iloc[p]-z1),norm(iris.iloc[p]-z2),norm(iris.iloc[p]-z3)]
    cluster.append((np.argmin(distances)+1))
iris["cluster"] = cluster
```

```
counter = 0
P = len(iris)
for p in range(P):
    if iris.iloc[p].cluster == y.iloc[p]:
        counter += 1
acc = counter / P
print("acc: " + str(acc))
```

```
acc: 0.2
```

Figure 3.3.10: Iris dataset's accuracy score

4.Conclusion

The Whale Optimization Algorithm and The Firefly Algorithm are one of the latest artificial intelligences algorithms developed. Like other metaheuristic algorithms, they were inspired by nature. While the firefly algorithm was influenced by the lights of fireflies, the whale algorithm was influenced by the hunting methods of humpback whales from air bubbles. In this study, performances and accuracy results of Firefly Algorithm and Whale Optimization Algorithm in data clustering were tried to be compared. Considering the increasing data day by day, the value of the work done in the field of optimization is also increasing. Although both of optimization algorithms works quickly with optimized results, we can see that the success rate we obtained from the Firefly Algorithm applying the cluster operation in this study is higher than the accuracy rate we obtained from the Whale Optimization Algorithm. To sum up, optimizing and running our data can provide many advantages for our many problem solutions by providing maximum efficiency in minimum time.

5.References

[1] Guido van Rossum and the Python development team, Python Tutorial, 2020.

[2] jupyter.org

[3] www.anaconda.com

[4] en.wikipedia.org/wiki/Iris_flower_data_set

[5] R. A. Fisher (1936). "The use of multiple measurements in taxonomic problems".

[6] archive.ics.uci.edu/ml/datasets/wine

[7] Hamza Aydemir, 2020, Optimizasyon Algoritmaları: Yarasa Algoritması

[8] Michael F. Lohrer, 2020, A Comparison Between the Firefly Algorithm and Particle Swarm Optimization

[9] S. Luke, Essentials of Metaheuristics, Lulu, 2009.

[10] X. S. Yang, Nature-Inspired Metaheuristic Algorithms, Luniver Press, 2008.

[11] X. S. Yang, "Firefly Algorithms for Multimodal Optimization," in Stochastic algorithms:

foundations and applications, 2009.

[12] Altherwi, A. (2020), Application of the Firefly Algorithm for Optimal Production and Demand Forecasting at Selected Industrial Plant

[13] Mirjalili, S., Lewis, A., The Whale Optimization Algorithm., Advances in Engineering Software, 95(2016)

[14] Goldbogen, J.A., Friedlaender, A.S., Calambokidis, J., Mckenna, M.F., Simon M, Nowacek, D.P., Integrative Approaches to the Study of Baleen Whale Diving Behavior, Feeding Performance, and Foraging Ecology, BioScience, 63(2013)

[15] Murat CANAYAZ, Recep ÖZDAĞ, 2017, DATA CLUSTERING BASED ON THE WHALE OPTIMIZATION

6.Resume

Name Surname: Begüm BOLAT

Birth Year: 12/09/1998

Phone Number: +90530 651 3207

E-Mail Address: bolatbegum98@gmail.com

6.1 – Education

2016 – Erdem Bayazıt Anatolian High School

2017 – Çukurova University, Computer Engineering