*A Mini-project report on*

**Two Cooperative ant colonies for feature selection using fuzzy models**

*Submitted in partial fulfilment for the award of the degree of*

**B.Tech**

*By*

1. DASARI SHIVA – 20BCE7075

2. DASARI SRIKANTH – 20BCE7076

3. BOJJA DHANESH – 20BCD7243

4. M. CHIRANJEEVI – 20BCE7375

5.MUHAMMAD SAHIL – 20BCE7615

6.M. BHANU PRAKASH – 20BCD7076

**COURSE NAME**

**Soft Computing (CSE2009)**



**School of computer science engineering**

**May 2023**

**CONTENTS**

**INTRODUCTION**

1.1 INTRODUCTION

1.2 LITERATURE SURVEY

1.3 PROBLEM STATEMENT

1.4 DETAILS OF DATASET

1.5 TECHNIQUE

1.5 WORKING PRINCIPLE

1.6 CHOICE OF TOOL BOX

1.7 FLOWCHART OF METHODOLOGY

1.8 IMPLEMENTATION

1.9 RESULTS AND DISCUSSIONS

1.10 SCREENSHOTS

1.11 REFERENCES

**ABSTRACT:**

In real-world databases, the number of potential features that are available is occasionally very wide, and it may be necessary to identify a limited subset for classification needs. Feature selection is one of the most crucial methods in data pre-processing for classification. decreasingly significant or strongly connected aspects. The objective is to identify a condensed set of features that exhibits the best classification accuracy for a classifier. Numerical data can be utilized to create rule-based fuzzy models, which can then be used as classifiers. This work uses fuzzy models as classifiers since rule-based frameworks have proven to be an effective qualitative description for classification systems. This study suggests a feature selection technique based on two cooperative ant colonies that minimizes two goals: the number of features and the classification error. For these goals, two pheromone matrices and two separate strategies are applied. When compared to other features selection approaches, the method performs as well as or better.

**Keywords:** Feature selection, Ant colony optimization, fuzzy modelling

**INTRODUCTION**

**1.1 INTRODUCTION:**

Feature selection is an important step in machine learning, which involves selecting a subset of relevant features from a dataset that will be used to build a predictive model. The goal of feature selection is to improve the accuracy and efficiency of the model by reducing the number of irrelevant and redundant features. However, the feature selection process can be challenging and time-consuming, especially when dealing with high-dimensional datasets.

In recent years, researchers have turned to bio-inspired algorithms for feature selection, with the goal of achieving better performance and faster convergence. One such algorithm is the ant colony optimization (ACO) algorithm, which is a metaheuristic inspired by the behaviour of real ant colonies. ACO algorithms have been successfully applied in many optimization problems, including feature selection.

This project aims to use fuzzy models and two cooperative ant colonies to select the most relevant features from a dataset. Fuzzy models are used to represent uncertainty and vagueness in data, making them well-suited for dealing with complex and uncertain datasets. The two cooperative ant colonies work together to explore the feature space and select the most relevant features based on a fitness function. The fitness function is determined by the accuracy of the fuzzy model, which is evaluated using a testing set.

The use of two cooperative ant colonies allows for a more efficient and effective feature selection process, as the colonies work together to explore the feature space and avoid getting stuck in local optima. By using fuzzy models and ACO algorithms, this project aims to improve the accuracy and efficiency of feature selection and provide a better understanding of how these algorithms can be applied in real-world problems.

**1.2 LITERATURE SURVEY:**

In this project a specific topic is covered that is “**How can two cooperative ant colonies be used for the feature selection using fuzzy models**?”. This project provides examples of how ant colony optimization, fuzzy logic, and other techniques can be combined to perform feature selection in data classification. They also provide a basis for further research on how cooperative ant colonies can be used for feature selection using fuzzy models.

**1.3 PROBLEM STATEMENT:**

To develop a two cooperative ant colonies algorithm that incorporates fuzzy models to evaluate the quality of the feature subsets for feature selection in machine learning. The algorithm aims to identify the optimal subset of features that maximize the accuracy and efficiency of the machine learning model.

**1.4 DETAILS OF DATASET:**

1. Breast Cancer dataset:

This database of breast cancer patients was acquired from the SEER Program of the NCI's November 2017 update, which offers details on population-based cancer statistics.

The dataset included female patients who had been diagnosed between 2006 and 2010 with infiltrating ductal and lobular carcinoma breast cancer (SEER primary cites recode NOS histology codes 8522/3). In the end, 4024 patients were included after the exclusion of patients with uncertain tumour sizes, studied regional LNs, positive regional LNs, and patients whose survival months were less than one month.

1. Sonar dataset:

The "sonar.mines" file contains 111 patterns that were produced by reflecting sonar sounds off a metal cylinder at different angles and under different circumstances. In the file"sonar.rocks,"97 patterns were collected from rocks under comparable circumstances. A rising-frequency frequency-modulated chirp characterises the transmitted sonar signal. Signals collected from numerous various aspect angles, ranging from 90 degrees for the cylinder to 180 degrees for the rock, are included in the data set.

Each pattern consists of a collection of 60 numbers between 0.0 and 1.0. Each value is an integration of the energy inside a specific frequency range over a specific amount of time. Since the frequencies are conveyed later during the chirp, the integration aperture for higher frequencies occurs later in time.

1. Wine dataset: The wine data set, which is a commonly used classification data and is published in the machine learning repository, covers the chemical analysis of 178 wines produced from the three different cultivars in the same Italian region. There are thirteen continuous qualities that can be classified: Malic acid, ethanol, ash, ash alkalinity, magnesium, total Proanthocyanism, phenols, flavanoids, nonflavanoids phenols, the OD280/OD315 of diluted wines, and proline.

**1.5 TECHNIQUE:**

Fuzzy logic, is a mathematical framework for dealing with uncertainty and imprecision. Fuzzy logic allows for reasoning with linguistic terms and degrees of membership in a fuzzy set. Fuzzy logic can be used in combination with other techniques, such as ACO, to provide a more flexible and robust approach to solving complex problems.

In the context of feature selection using two cooperative ant colonies, the ACO algorithm is used to search for the most informative features in a dataset. The algorithm uses two ant colonies to explore both global and local search spaces. The selected features are then further refined using fuzzy logic to improve the accuracy of the classification model.

Overall, the combination of ACO and fuzzy logic allows for an effective and efficient approach to feature selection in classification problems.

**1.6 WORKING PRINCIPLE:**

1. Initialization: The algorithm initializes two ant colonies, one for global search and another for local search. Each ant colony is composed of a population of artificial ants that construct solutions to the feature selection problem.
2. Solution Construction: Each ant in the global search colony constructs a solution by selecting a subset of features from the entire feature space. Similarly, each ant in the local search colony constructs a solution by selecting a subset of features from a neighborhood around a randomly selected feature.
3. Pheromone Update: After constructing a solution, each ant updates the pheromone trail on the features it has selected. The pheromone levels are updated based on the quality of the solution, such that features selected in good solutions receive higher pheromone levels.
4. Feature Selection: After a fixed number of iterations, the ant colonies stop constructing new solutions, and the features with the highest pheromone levels are selected as the most informative features.
5. Fuzzy Logic: The selected features are further refined using fuzzy logic. Fuzzy models are constructed using the selected features and a fuzzy inference system is used to classify new instances. The fuzzy inference system allows for reasoning with uncertain and imprecise data, providing a more robust approach to classification.
6. Evaluation: The performance of the feature selection algorithm is evaluated using a classification metric, such as accuracy or AUC. The algorithm is compared to other feature selection techniques to determine its effectiveness.

Overall, the working principle of two cooperative ant colonies for feature selection using fuzzy models combines the exploration capabilities of ant colony optimization with the flexibility of fuzzy logic to provide an effective and efficient approach to feature selection in classification problems.

**1.7 TOOL BOX USED:**

**MATLAB:** MATLAB is a popular tool for implementing fuzzy logic and ant colony optimization algorithms. MATLAB provides built-in functions and tools for developing and testing fuzzy models, as well as optimization algorithms such as ant colony optimization. The Fuzzy Logic Toolbox and the Global Optimization Toolbox are two MATLAB toolboxes that can be used for implementing two cooperative ant colonies for feature selection.

**1.8 FLOWCHART OF METHODOLOGY:**

Train Data

Insert Data

Select Feature by Ant Colony

Select Feature by Ant Colony

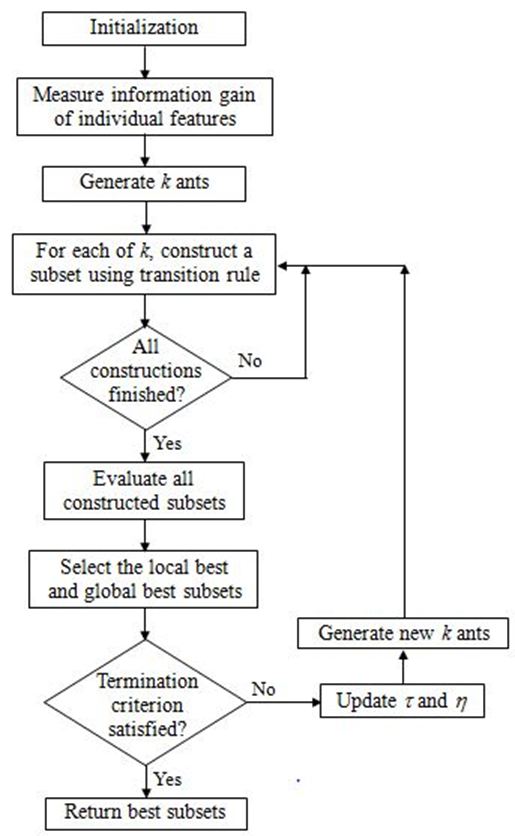
Calculate Performance

Classify the Data

Match Data

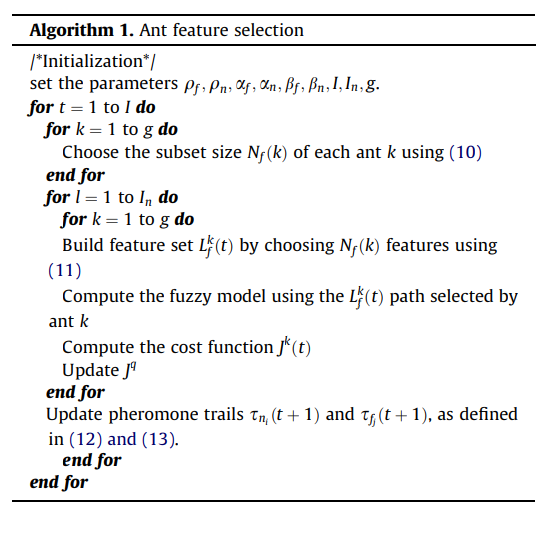
Trained Data

Test Data

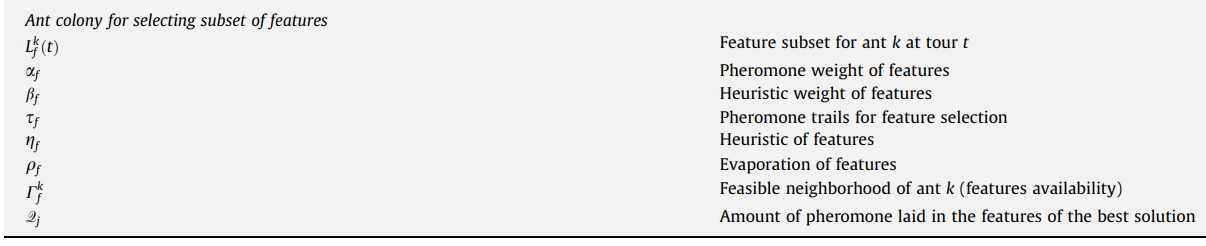
**A New Hybrid ACO-based Feature Selection Algorithm-ACOFS**

**1.9 IMPLEMENTATION:**

**Algorithm:**



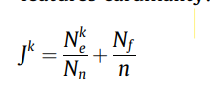
**Variables used in the algorithm:**



**Proposed Ant Feature Selection:**

1. **Proposed Algorithm:**

The objective function of this optimization algorithm aggregate both criteria, the minimization of the classification error rate and the minimization of the features cardinality:

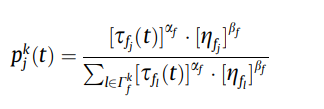


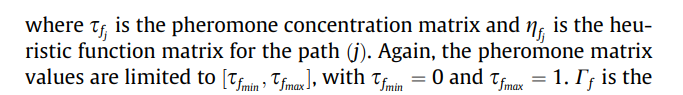
Where Nn is the number of used data samples, n is the total number of features and k is a given ant.

1. **Probabilistic Rule:**

Consider a problem with Nf nodes and two colonies of g ants. First, g ants of the first colony randomly select the number of nodes Nf to be used by the g ants of the second colony. The probability that an ant k chooses the features cardinality NfðkÞ is given by:

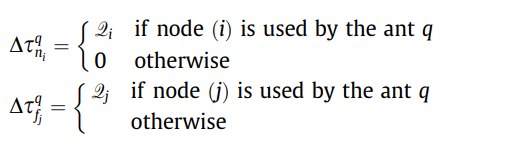
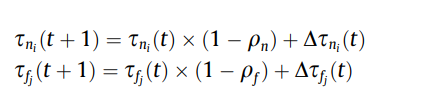
The probability that an ant k chooses feature j as the next feature to visit is given by:



Feasible neighborhood of k

1. **Updating Rule:**

After a complete tour, when all the g ants have visited all the Nf(k) nodes, both pheromone concentration in the trails are updated by:



The number of nodes Nf(k) that each ant k has to visit on each tour t is only updated every In tours (iterations), in order to allow the search for the best features for each features cardinality Nf.

**CODE:**

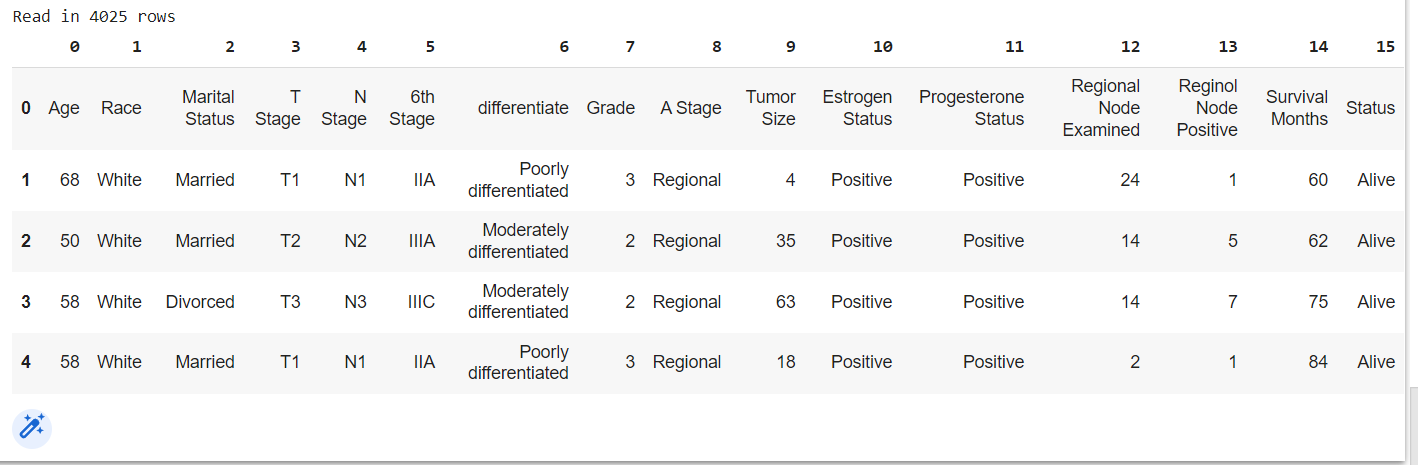
CODE:(GENERAL CLASSIFICATION):

import pandas as pd

df = pd.read\_csv("/content/sample\_data/Breast\_Cancer.csv", header=None) #file contains no header info

print(f"Read in {len(df)} rows")

df.head()



import pandas as pd

df = pd.read\_csv("/content/sample\_data/Breast\_Cancer.csv", header=None) #file contains no header info

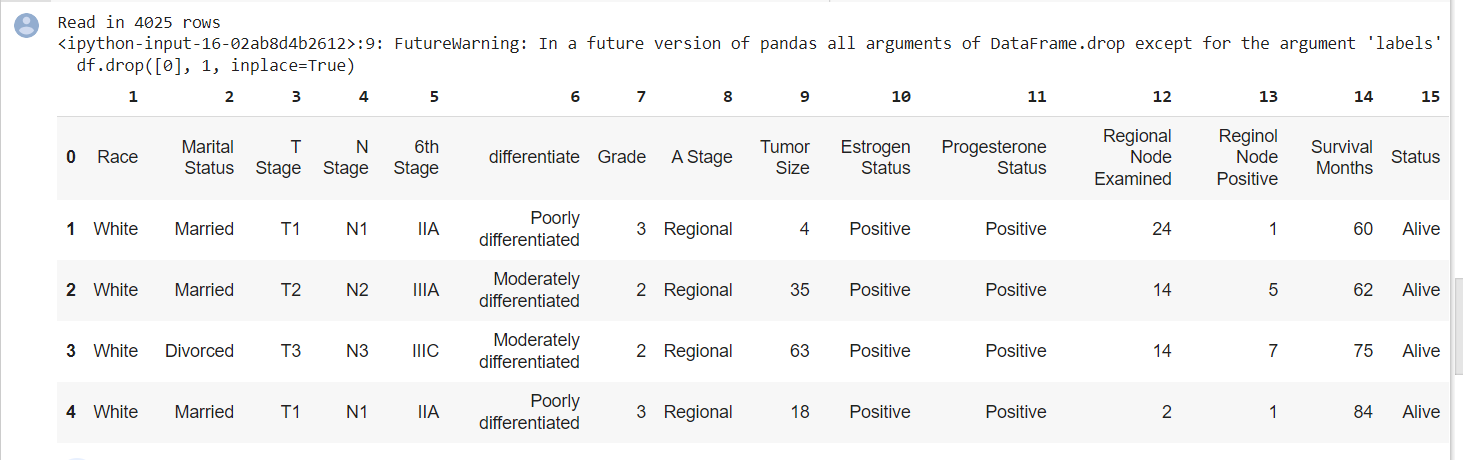
print(f"Read in {len(df)} rows")

df.head()

df.replace("?", 10000, inplace=True) #10,000 is way beyond the range of columns provided so acts as an outlier

df.drop([0], 1, inplace=True)

df.head()



**Final code and output:**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.gaussian\_process import GaussianProcessClassifier

from sklearn.gaussian\_process.kernels import RBF

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.discriminant\_analysis import QuadraticDiscriminantAnalysis

# Read in data

url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data'

df = pd.read\_csv(url, header=None)

# Replace "?" with 10000

df.replace("?", 10000, inplace=True)

# Drop first column and create X and y

X = np.array(df.drop([0, 10], axis=1))

y = np.array(df[10])

# Split data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=43)

# Set up classifiers

names = ["Logistic Regression", "Nearest Neighbors", "Linear SVM", "RBF SVM", "Gaussian Process",

         "Decision Tree", "Random Forest", "Neural Net", "AdaBoost", "Naive Bayes", "QDA"]

classifiers = [

    LogisticRegression(max\_iter=300),

    KNeighborsClassifier(),

    SVC(kernel="linear", C=0.025),

    SVC(gamma=2, C=1),

    GaussianProcessClassifier(1.0 \* RBF(1.0)),

    DecisionTreeClassifier(max\_depth=5, random\_state=43),

    RandomForestClassifier(max\_depth=5, random\_state=43),

    MLPClassifier(alpha=1, max\_iter=1000),

    AdaBoostClassifier(),

    GaussianNB(),

    QuadraticDiscriminantAnalysis()]

# Train and evaluate classifiers

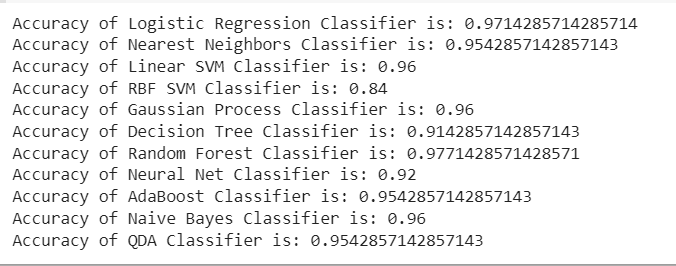
for name, clf in zip(names, classifiers):

    clf.fit(X\_train, y\_train)

    score = clf.score(X\_test, y\_test)

    print(f"Accuracy of {name} Classifier is: {score}")

**OUTPUT:**

****

**b**

**ACO algorithm:**

import numpy as np

import matplotlib.pyplot as plt

import sys

IN\_COLAB = 'google.colab' in sys.modules

class Catagorization:

    def \_\_init\_\_(self, num\_ants, num\_rules\_converge, data, attributes, classes):

        self.num\_ants = num\_ants

        self.num\_rules\_converge = num\_rules\_converge

        self.data = data

        self.original\_data = data.copy()

        self.attributes = attributes

        self.classes = classes

        self.heuristic = self.calc\_heuristic()

        self.pharamones = self.init\_pharamones()

        self.discovered\_rules = []

        self.qualities = [[]]

        self.leftover\_cases = 35

        self.init\_ants()

    def init\_ants(self):

        min\_cases = 5

        self.ants = [Ant(self.pharamones, self.heuristic, self.attributes, min\_cases, self.classes) for i in range(self.num\_ants)]

    def run\_simulation(self, verbose = False, supress\_output = False):

        while len(self.data) > self.leftover\_cases:

            self.converged\_rules = 1

            best\_rule = []

            best\_quality = -1

            consequent = None

            for j, ant in enumerate(self.ants):

                self.run\_one\_ant(ant)

                if best\_quality < 0 and set(self.ants[j-1].rule) == set(ant.rule) and ant.rule:

                    self.converged\_rules += 1

                else:

                    self.converged\_rules = 1

                if self.converged\_rules == self.num\_rules\_converge:

                    print('rules have converged')

                    break

                if ant.quality > best\_quality:

                    best\_quality = ant.quality

                    best\_rule = ant.rule

                    consequent = ant.consequent

            n = self.remove\_relevant\_cases(best\_rule, consequent)

            assert(not self.find\_relevant\_cases(best\_rule))

            self.discovered\_rules.append((best\_rule, consequent, best\_quality, n))

            self.pharamones = self.init\_pharamones()

            self.init\_ants()

            self.qualities.append([])

            if verbose: print('remaining data: ', len(self.data))

        if not supress\_output: print('simulation done. Remaining cases: ', len(self.data))

    def run\_one\_ant(self, ant):

        ant.pharamones = self.pharamones

        ant.add\_terms(self.data)

        q = self.prune\_ant(ant)

        consequent = self.calc\_consequent(ant.rule)

        self.update\_pharamones(ant.rule, q)

        ant.quality = q

        ant.consequent = consequent

        self.qualities[-1].append(q)

    def calc\_heuristic(self):

        probs = {}

        heuristic = {}

        for index, i in enumerate(self.attributes.keys()):

            for j in self.attributes[i]:

                for k in self.classes:

                    a = [game for game in self.data if k == game[-1]]

                    b = [game for game in a if j == game[index]]

                    p = len(b)/len(self.data)

                    c = probs.get((index,j),[])

                    c.append(np.log2(p\*\*p))

                    probs[(index,j)] = c

                heuristic[(index,j)] = -sum(probs[(index,j)])

        return heuristic

    def pick\_best\_rule(self):

        most\_correct = 0

        best\_ant = None

        for ant in self.ants:

            consequence, num\_correct = self.calc\_num\_correct(ant.rule)

            ant.consequence = consequence

            if num\_correct > most\_correct:

                most\_correct = num\_correct

                best\_ant = ant

        return most\_correct, best\_ant

    def find\_relevant\_cases(self, rule):

        relevant\_cases = self.data.copy()

        for case in self.data:

            for term in rule:

                if case[term[0]] != term[1]:

                    relevant\_cases.remove(case)

                    break

        return relevant\_cases

    def remove\_relevant\_cases(self, rule, consequent):

        relevant\_cases = self.find\_relevant\_cases(rule)

        count = 0

        for case in relevant\_cases:

            self.data.remove(case)

            if case[-1] == consequent:

                count += 1

        return count

    def calc\_consequent(self, rule):

        relevant\_cases = self.find\_relevant\_cases(rule)

        classes = {}

        if not relevant\_cases:

            relevant\_cases = self.data

        for case in relevant\_cases:

            classes[str(case[-1])] = classes.get(str(case[-1]), 0) + 1

        res = max(classes, key=classes.get)

        return res

    def prune\_ant(self, ant):

        """Iteratively goes through the rulelist for an ant and sees which rules are not helping the quality of the rule"""

        n = len(ant.rule)

        if n == 0:

            return 0

        for iteration in range(n):

            max\_delta\_quality = 0

            best\_new\_rule = ant.rule

            consequent = self.calc\_consequent(ant.rule)

            base\_quality = self.calc\_quality(ant.rule, consequent)

            for i, term in enumerate(ant.rule):

                new\_rule = ant.rule[0:i] + ant.rule[i+1:]

                new\_consequent = self.calc\_consequent(new\_rule)

                new\_quality = self.calc\_quality(new\_rule, new\_consequent)

                if max\_delta\_quality <= new\_quality - base\_quality:

                    max\_delta\_quality = new\_quality - base\_quality

                    best\_new\_rule = new\_rule

            if max\_delta\_quality >= 0:

                if best\_new\_rule:

                    ant.rule = best\_new\_rule

                    max\_delta\_quality = 0

            else:

                break

        return max\_delta\_quality + base\_quality

    def calc\_quality(self, rule, consequent):

        relevant\_cases = self.find\_relevant\_cases(rule)

        num\_cases\_covered = len(relevant\_cases)

        num\_cases\_not\_covered = len(self.data) - len(relevant\_cases)

        true\_positives = len([case for case in relevant\_cases if case[-1] == consequent])

        false\_positives = len(relevant\_cases) - true\_positives

        false\_negatives = len([case for case in self.data if case not in relevant\_cases and case[-1] == consequent])

        true\_negatives = len(self.data) - len(relevant\_cases) - false\_negatives

        sensitivity = true\_positives / (true\_positives + false\_negatives)

        if true\_negatives == 0: return 0

        specificity = true\_negatives / (true\_negatives + false\_positives)

        return sensitivity \* specificity

    def init\_pharamones(self):

        pharamones = {}

        total = 0

        for attribute in self.attributes.keys():

            total += len(self.attributes[attribute])

        initial\_value = 1/total

        for index, i in enumerate(self.attributes.keys()):

            for j in self.attributes[i]:

                pharamones[(index,j)] = initial\_value

        return pharamones

    def update\_pharamones(self, rule, quality):

        for term in rule:

            self.pharamones[(term[0], term[1])] += quality

        normalization\_factor = sum(self.pharamones.values())

        keys = self.pharamones.keys()

        for key in keys:

            self.pharamones[key] /= normalization\_factor

        assert(sum(self.pharamones.values()) - 1 < 0.0001)

    def evaluate\_discovered\_rules(self, data):

        num\_correct = 0

        num\_total = len(data)

        self.data = data

        for rule in self.discovered\_rules:

            num\_correct += self.remove\_relevant\_cases(rule[0], rule[1])

        return num\_correct / num\_total

class Ant:

    def \_\_init\_\_(self,pharamones, heuristic, attributes, min\_cases\_per\_rule, classes):

        ""

        self.rule = []

        self.pharamones = pharamones

        self.heuristic = heuristic

        self.decision = np.zeros((9,3))

        self.k = 2

        self.classes = classes

        self.attributes = attributes

        self.min\_cases\_per\_rule = min\_cases\_per\_rule

        self.used\_attributes = []

    def add\_terms(self, data):

        "adds a term to the ruleset based on the pharamone trail and heuristic function"

        quit = False

        tries = 0

        for i in self.attributes.keys():

            probs = self.calc\_prob()

            picking = True

            while picking:

                term = self.pick\_term(probs)

                if len(self.find\_relevant\_cases(self.rule+[term], data)) > self.min\_cases\_per\_rule:

                    if term[0] not in self.used\_attributes:

                        self.used\_attributes.append(term[0])

                        picking = False

                else:

                    tries += 1

                if tries > 10 and self.rule:

                    picking = False

                    quit = True

                if tries > 25:

                    quit = True

            if not quit:

                self.rule.append(term)

            else:

                break

    def normalize(self, function):

        norm = {}

        for index, i in enumerate(self.attributes.keys()):

            for j in self.attributes[i]:

                num = function(index, j)

                if num == 0:

                    norm[(index,j)] = 0

                    continue

                unused\_attributes = len(self.attributes.keys())-len(self.rule)

                normalization\_factor = 0

                for jj in self.attributes[i]:

                    normalization\_factor += function(index, jj)

                norm[(index,j)] = num / (unused\_attributes \* normalization\_factor)

        return norm

    def normalize\_heuristic(self):

        def f(i, j):

            return np.log2(self.k) - self.heuristic[(i,j)]

        return self.normalize(f)

    def calc\_prob(self):

        self.normalized\_heuristic = self.normalize\_heuristic()

        def f(i, j):

            if i not in self.used\_attributes:

                return self.pharamones[(i,j)] \* self.normalized\_heuristic[(i,j)]

            else:

                return 0

        return self.normalize(f)

    def pick\_term(self, probs):

        index = np.random.choice(len(probs), 1, p = list(probs.values()))

        index = index[0]

        term = list(probs.keys())[index]

        return term

    def find\_relevant\_cases(self, rule, data):

        relevant\_cases = data.copy()

        for case in data:

            for term in rule:

                if case[term[0]] != term[1]:

                    relevant\_cases.remove(case)

                    break

        return relevant\_cases

import os

import string

def get\_train\_and\_test\_from\_file(url, file, special = None):

    !wget -cq $url

    data = open(file, 'r')

    data\_listed = []

    for i in data.readlines():

        if not special:

            data\_listed.append(tuple(str.split(i[:-1], ',')))

        elif special == 'breast':

            line = str.split(i[:-1], ',')

            c = line.pop(0)

            line.append(c)

            data\_listed.append(tuple(line))

        elif special == 'wisconsin':

            data\_listed.append(tuple(str.split(i[:-1], ',')[1:]))

    n=len(data\_listed)

    train\_ind = np.random.choice(n, round(n\*0.8), replace=False)

    train = [data\_listed[index] for index in train\_ind]

    test = [case for case in data\_listed if case not in train]

    return train, test

"""### Running the Miner on Breast Cancer Data (Wisconsin)"""

# Number of times to run the miner

num\_runs = 1

# Number of ants to use

num\_ants = 50

accuracies = []

num\_rules = []

for i in range(num\_runs):

    train, test = get\_train\_and\_test\_from\_file('/content/sample\_data/Breast\_Cancer.csv', '/content/sample\_data/Breast\_Cancer.csv', 'wisconsin')

    attribute\_list = ['Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape', 'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin', 'Normal Nucleoli', 'Mitoses']

    attributes = {}

    for attribute in attribute\_list:

        attributes[attribute] = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']

    classes = ['2', '4']

    s = Catagorization(num\_ants, 10, train, attributes, classes)

    s.run\_simulation()

    accuracies.append(s.evaluate\_discovered\_rules(test))

    num\_rules.append(len(s.discovered\_rules))

print('Average Accuracy is: ', np.mean(accuracies))

print('Average number of features per ruleset is: ', np.mean(num\_rules))

print('Number of rules per ruleset std. is', np.mean(num\_rules))

"""### Sweeping number of Ants

"""

# Number of times to run the miner

num\_runs = 15

# Number of ants to use

num\_ants\_vec = [5, 10, 15, 20, 25]

all\_accuracies = []

all\_num\_rules = []

all\_num\_terms = []

for num\_ants in num\_ants\_vec:

    all\_accuracies.append([])

    all\_num\_rules.append([])

    all\_num\_terms.append([])

    for i in range(num\_runs):

        train, test = get\_train\_and\_test\_from\_file('/content/sample\_data/Breast\_Cancer.csv', '/content/sample\_data/Breast\_Cancer.csv', 'wisconsin')

        attribute\_list = ['Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape', 'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei', 'Bland Chromatin', 'Normal Nucleoli', 'Mitoses']

        attributes = {}

        for attribute in attribute\_list:

            attributes[attribute] = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']

        classes = ['2', '4']

        s = Catagorization(num\_ants, 10, train, attributes, classes)

        s.run\_simulation(supress\_output = True)

        all\_accuracies[-1].append(s.evaluate\_discovered\_rules(test))

        all\_num\_rules[-1].append(len(s.discovered\_rules))

        all\_num\_terms[-1].append(np.mean([len(rule[1]) for rule in s.discovered\_rules]))

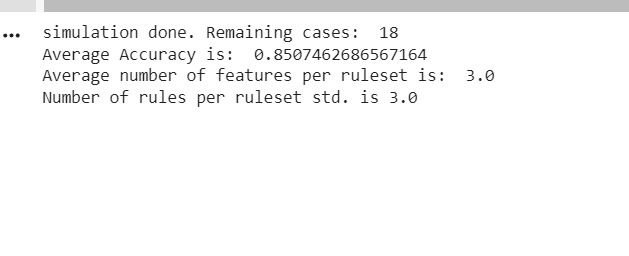
plt.boxplot(all\_accuracies, positions=[n/5 for n in num\_ants\_vec])

plt.title('Accuracy vs Number of Ants Used (Wisconsin Breast Cancer)')

plt.xlabel('Number of Ants Used (x5)')

plt.ylabel('Accuracy of Ruleset')

plt.show()



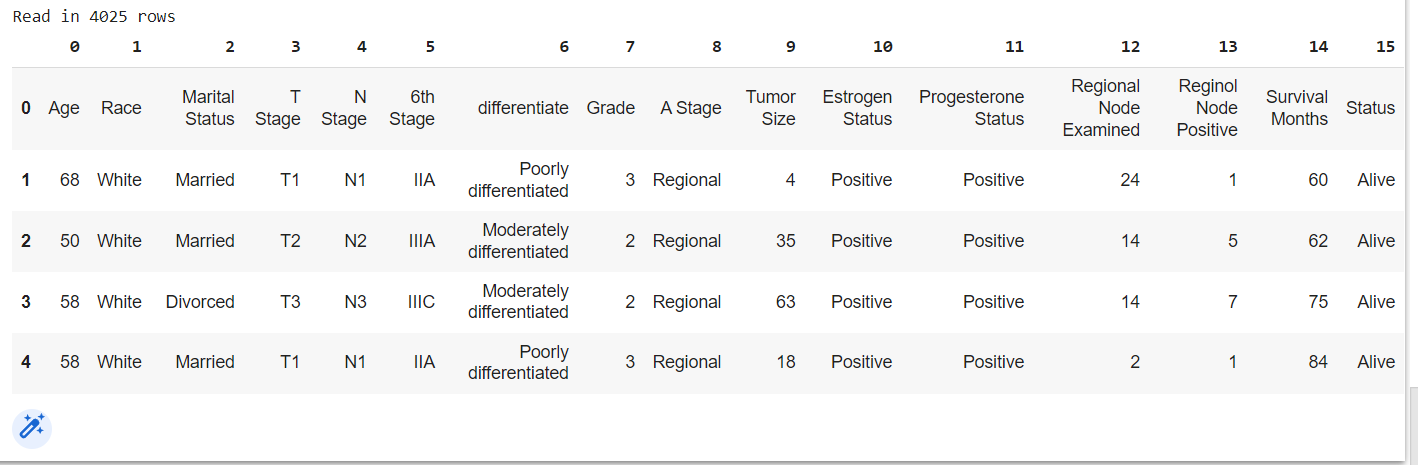
**1.10 RESULTS AND DISCUSSIONS:**

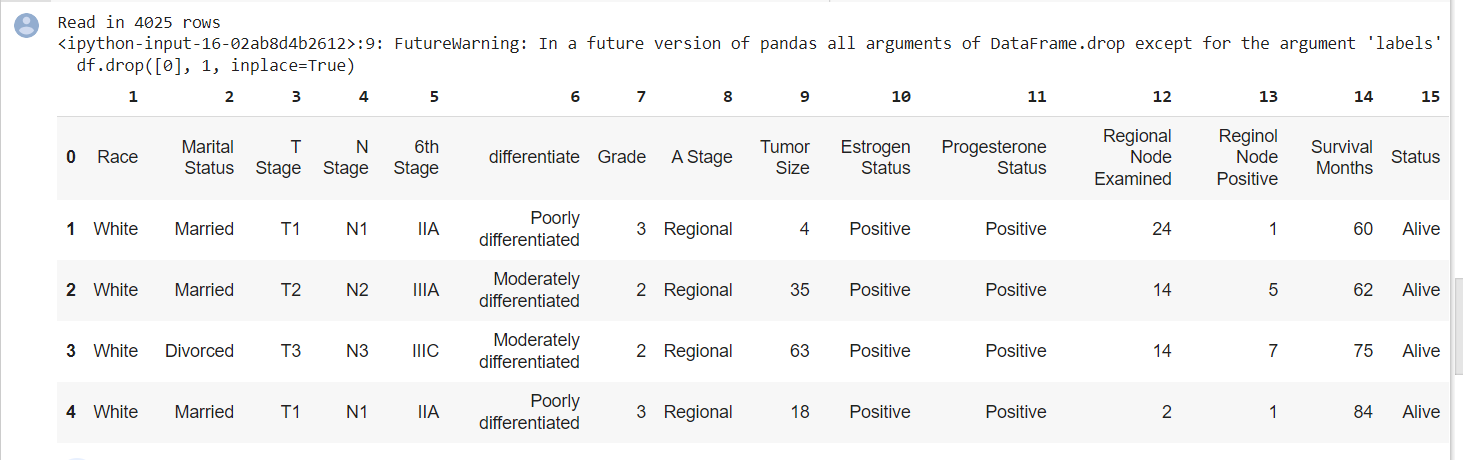
A feature selection algorithm using two cooperative ant colonies was developed in this project. Our main intention of this project is to selecting the most relevant features from the more number of features from a dataset. feature selection algorithm uses fuzzy classifiers. we have applied this algorithm on breast cancer dataset. The ant based Classification algorithm made a very best relevant features to reach the target in this project. Finally, best 3 features are selected using the Ant colony feature selection algorithm.

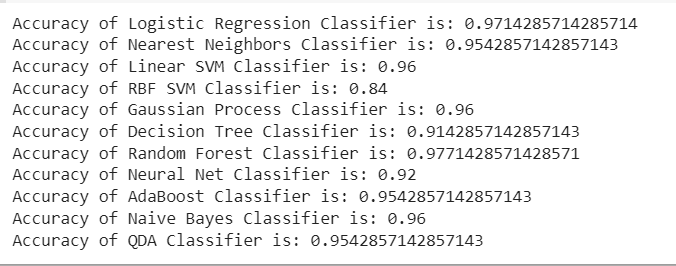
The feature selection proposed in this project was applied on Breast cancer dataset. This selection algorithm selects the strongly relevant features among all the irrelevant features from the dataset.

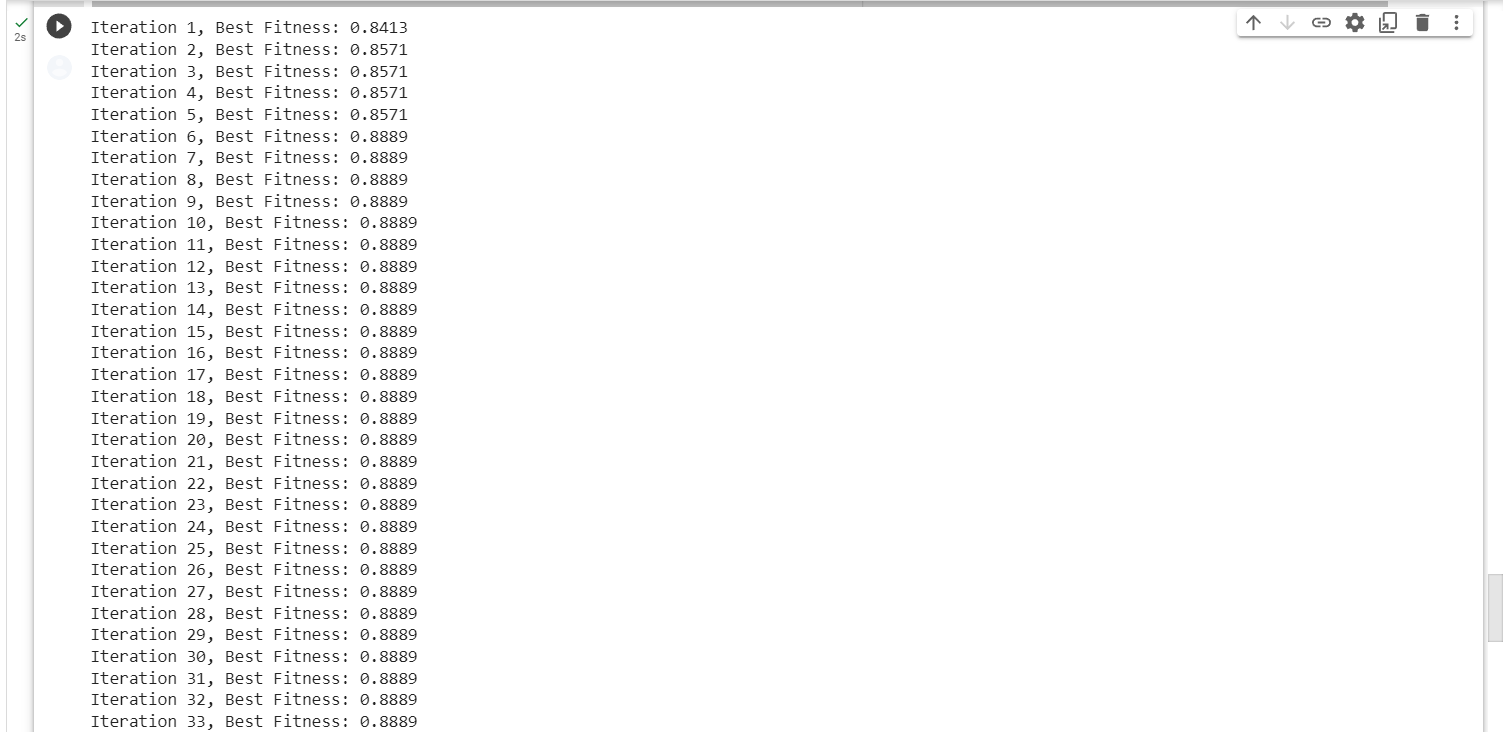
In Future this project can be developed so that relevant features can be selected from a very huge dataset and this algorithm can be combined with neural networks to make this much more better in classifying the features very easily and effectively.

**1.11 SCREENSHOTS:**





****





**1.12 REFERENCES**

1. <https://www.sciencedirect.com/science/article/abs/pii/S0957417409007593>
2. <https://www.researchgate.net/publication/220215664_Two_cooperative_ant_colonies_for_feature_selection_using_fuzzy_models>
3. <https://dl.acm.org/doi/10.1016/j.eswa.2009.08.026>
4. <https://link.springer.com/chapter/10.1007/978-3-642-13935-2_16>
5. <https://www.intechopen.com/chapters/39688>