A PRELIMINARY REPORT ON

Classifying the Iris dataset with Fuzzy logic and Genetic Algorithm

BACHELOR OF ENGINEERING (Computer Engineering)

BY

Ashay Koradia	41429
Nachiket Erlekar	41434

Under The Guidance of

Prof. U. S. Pawar

DEPARTMENT OF COMPUTER ENGINEERING

Pune Institute of Computer Technology

Dhankawadi, Pune-411046

SAVITRIBAI PHULE PUNE UNIVERSITY

2020 -2021

Problem Statement:

Classifying the Iris dataset with Fuzzy logic and Genetic Algorithm.

Abstract :

The use of fuzzy logic to describe abstract concepts and when designing decision making systems to make decisions that are much closer to the way a human does is an interesting and useful area to explore. To effectively implement these types of systems expert knowledge is required in the application area. This can be a problem if the people designing a system of this nature are not experts in the application area since it is not always possible to have an expert working continuously with a project. This problem can be alleviated by using data supplied by an expert to let the system learn from it and thereby improve the system without continuous input from experts.

Index :

Introduction :

In recent years the use of fuzzy logic in fuzzy systems has been implemented with good success in many different types of systems ranging from controlling airplanes to sake brewing. As such, the knowledge on how to construct systems using fuzzy logic to solve problems is a valuable one.

This project includes an implementation of a fuzzy system that will be used to classify the species of the iris flower based on the iris data set. This is a well known benchmark problem commonly used in machine learning research. To improve the accuracy of the classifier a genetic algorithm (GA) is added to the implementation to fine tune the membership functions.

As will be seen the use of GA can be implemented with a noticeable improvement in the accuracy of the fuzzy classifier.

Theory:

Many common applications of machine learning, from customer targeting to medical diagnosis, arise from complex relationships between features (also-called input variables or characteristics).Feature selection, or inputs selection, is the process of finding the most relevant inputs for a predictive model. These techniques can be used to identify and remove unneeded, irrelevant and redundant features that do not contribute or decrease the accuracy of the predictive model.

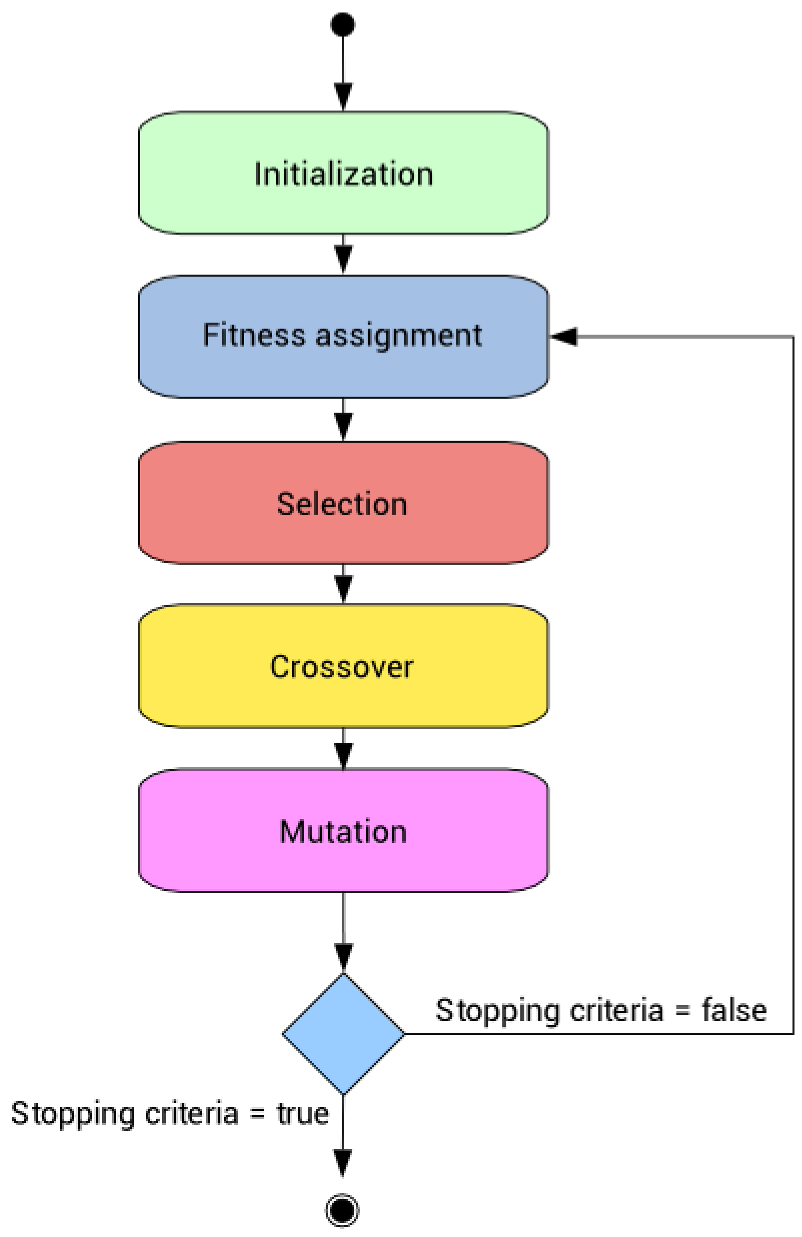
Mathematically, inputs selection is formulated as a combinatorial optimization problem. Here the function to optimize is the generalization performance of the predictive model, represented by the error term on the selection instances of a data set. The design variables are the inclusion (1) or the exclusion (0) of the input variables in the neural network.

An exhaustive selection of features would evaluate lots of different combinations (2^N, where N is the number of features). This process requires lots of computational work and, if the number of features is big, becomes impracticable. Therefore, we need intelligent methods that allow the selection of features in practice.

One of the most advanced algorithms for feature selection is the genetic algorithm. This is a stochastic method for function optimization based on the mechanics of natural genetics and biological evolution. In this article we show how genetic algorithms can be applied to optimize the performance of a predictive model, by selecting the most relevant features.In nature, the genes of organisms tend to evolve over successive generations to better adapt to the environment. The genetic algorithm is an heuristic optimization method inspired by that procedure of natural evolution.

Genetic algorithms operate on a population of individuals to produce better and better approximations. At each generation, a new population is created by the process of selecting individuals according to their level of fitness in the problem domain, and recombining them together using operators borrowed from natural genetics. The offspring might also undergo mutation.

Design / Implementation :



In our case, each individual in the population represents a predictive model. The number of genes is the total number of features in the data set. Genes here are binary values, and represent the inclusion or not of particular features in the model. The number of individuals, or population size, must be chosen for each application. Usually, this is set to be 10N, being N the number of features.

Evolutionary algorithms have three main characteristics:

1. **Population-Based**: Evolutionary algorithms are to optimize a process in which current solutions are bad to generate new better solutions. The set of current solutions from which new solutions are to be generated is called the population.
2. **Fitness-Oriented**: If there are several solutions, how to say that one solution is better than another? There is a fitness value associated with each individual solution calculated from a fitness function. Such fitness value reflects how good the solution is.
3. **Variation-Driven**: If there is no acceptable solution in the current population according to the fitness function calculated from each individual, we should make something to generate new better solutions. As a result, individual solutions will undergo a number of variations to generate new solutions.

Source Code :

genetic_algorithm.py

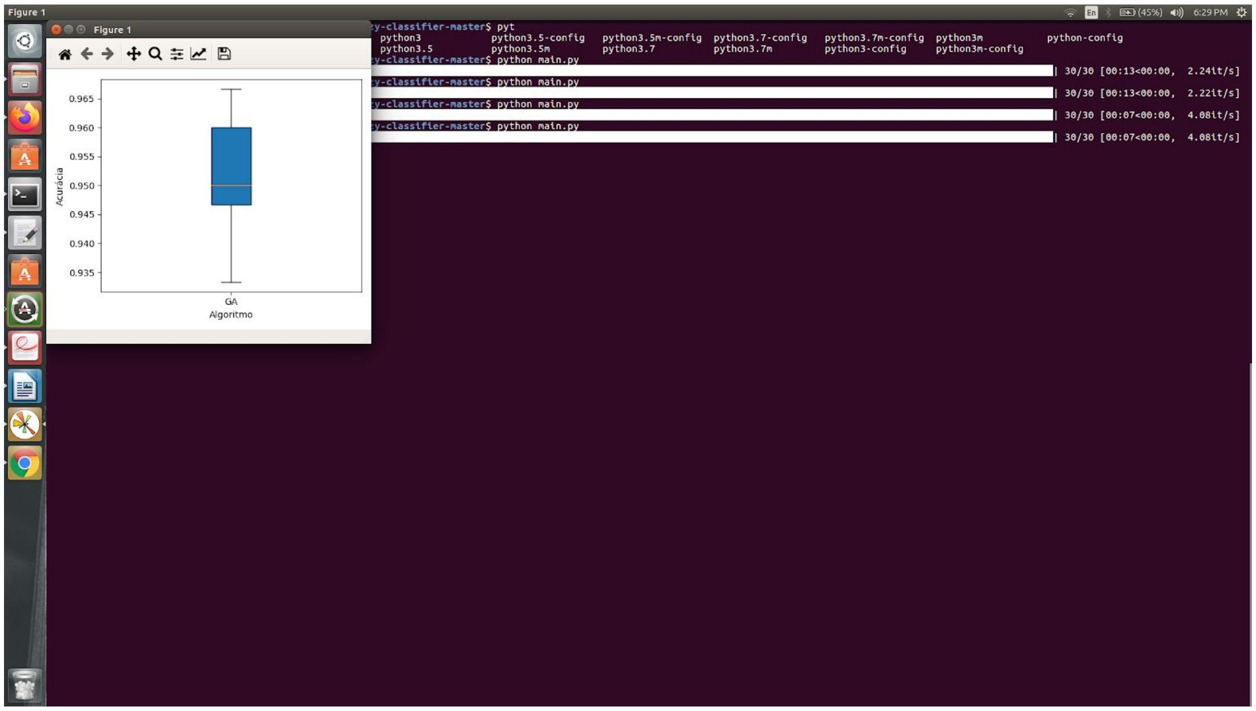
```
import numpy as np

def _best(population, fitness_func, best, fbest):
    # best, fbest = None, None
    for i in range(population[0].shape[0]):
        if population[1][i] > -1.0:
            tmp = population[1][i]
        else:
            tmp = fitness_func(population[0][i])
            population[1][i] = tmp
        if best is None or tmp < fbest:
            best = population[0][i]
            fbest = tmp
        return best.copy(), fbest.copy()


def _tournament_selection(population, fitness_func):
    # k == 2
    idxs = np.random.permutation(np.arange(population[0].shape[0]))
    parent1 = population[0][idxs[0], :]
    parent2 = population[0][idxs[1], :]
    if population[1][idxs[0]] > -1.0:
        fitness1 = population[1][idxs[0]]
    else:
        fitness1 = fitness_func(parent1)

        population[1][idxs[0]] = fitness1

    if population[1][idxs[1]] > -1.0:

        fitness2 = population[1][idxs[1]]

    else:

        fitness2 = fitness_func(parent2)

        population[1][idxs[1]] = fitness2

    return parent1 if fitness1 < fitness2 else parent2



def _individuals(size):

        return np.random.rand(size)

def _mutate(individual):

    idx = np.random.randint(low=0, high=individual.shape[0])

    individual[idx] = np.random.rand()

    return individual


def _crossover(male, female, alpha=0.9):

    """

    BLX-alpha crossover

    """

    shift = np.abs(male - female) * alpha

    mmin = np.amin([male, female], axis=0) - shift

    mmax = np.amax([male, female], axis=0) + shift
    mmin = np.clip(mmin, 0, 1)

    mmax = np.clip(mmax, 0, 1)

    offspring1 = np.random.uniform(low=mmin, high=mmax)

    offspring2 = np.random.uniform(low=mmin, high=mmax)

    return offspring1, offspring2


def genetic_algorithm(fitness_func, dim, n_individuals=10, epochs=50,
crossover_rate=0.9, mutation_rate=0.1, verbose=False):

    assert n_individuals % 2 == 0

    population = [np.array([_individuals(dim) for _ in range(n_individuals)]),

            np.zeros(n_individuals) - 1.0]

            children = np.zeros((n_individuals, dim))

            best, fbest = None, None

            for e in range(epochs):

                    for c in range(0, n_individuals, 2):

                            parent1 = _tournament_selection(population, fitness_func)

                            parent2 = _tournament_selection(population, fitness_func)

                            while np.array_equal(parent1, parent2):

                                    parent2 = _tournament_selection(population,
fitness_func)

                            if np.random.uniform() < crossover_rate:

                                    offspring1, offspring2 = _crossover(parent1, parent2)

                                    children[c, :] = offspring1

                                    children[c+1, :] = offspring2

                            else:
```

```
                children[c, :] = parent1
                children[c+1, :] = parent2
            if np.random.uniform() < mutation_rate:
                children[c, :] = _mutate(children[c, :])
            if np.random.uniform() < mutation_rate:
                children[c+1, :] = _mutate(children[c+1, :])
        best, fbest = _best(population, fitness_func, best, fbest)
        population[0][:] = children[:]
        population[1][:] = -1.0
        children[:] = 0.0
        if verbose:
            print('epoch {:2d}, best fitness = {:.10f}'.format(e, fbest))

    return best, fbest
```

main.py

```
import matplotlib.pyplot as plt
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
from sklearn import datasets
from genetic_algorithm import genetic_algorithm
from tqdm import tqdm
import matplotlib.pyplot as plt
def normalize_dataset(dataset):
    # Normalize the dataset to [0, 1]
    min_arr = np.amin(dataset, axis=0)
    return (dataset - min_arr) / (np.amax(dataset, axis=0) - min_arr)


def evaluate_new_fuzzy_system(w1, w2, w3, w4, data, target):
        universe = np.linspace(0, 1, 100)
        x = []
        for w in [w1, w2, w3, w4]:
                x.append({'s': fuzz.trimf(universe, [0.0, 0.0, w]),
                          'm': fuzz.trimf(universe, [0.0, w, 1.0]),
                            'l': fuzz.trimf(universe, [w, 1.0, 1.0])})
        x_memb = []
        for i in range(4):
                x_memb.append({})
                for t in ['s', 'm', 'l']:
                        x_memb[i][t] = fuzz.interp_membership(universe, x[i][t], data[:, i])

        is_setosa = np.fmin(np.fmax(x_memb[2]['s'], x_memb[2]['m']), x_memb[3]['s'])
        is_versicolor = np.fmax(np.fmin(np.fmin(np.fmin(np.fmax(x_memb[0]['s'],
x_memb[0]['l']), np.fmax(x_memb[1]['m'], x_memb[1]['l'])), np.fmax(x_memb[2]['m'],
x_memb[2]['l'])),x_memb[3]['m']), np.fmin(x_memb[0]['m'],
np.fmin(np.fmin(np.fmax(x_memb[1]['s'], x_memb[1]['m']),x_memb[2]['s']),
x_memb[3]['l'])))
         is_virginica = np.fmin(np.fmin(np.fmax(x_memb[1]['s'], x_memb[1]['m']),
x_memb[2]['l']), x_memb[3]['l'])

        result = np.argmax([is_setosa, is_versicolor, is_virginica], axis=0)
        return (result == target).mean()



def main():
    iris = datasets.load_iris()
    normalized_iris = normalize_dataset(iris.data)
    n_features = normalized_iris.shape[1]
    fitness = lambda w: 1.0 - evaluate_new_fuzzy_system(w[0], w[1], w[2], w[3],
normalized_iris, iris.target)

     record = {'GA': []}
    for _ in tqdm(range(30)):
            best, fbest = genetic_algorithm(fitness_func=fitness, dim=n_features,
n_individuals=10, epochs=30, verbose=False)
            record['GA'].append(1.0 - fbest)

 fig, ax = plt.subplots(figsize=(5, 4))
 ax.boxplot(list(record.values()), vert=True, patch_artist=True, labels=list(record.keys()))

 ax.set_xlabel('Algoritmo')
 ax.set_ylabel('Accuracy')

  plt.tight_layout()
  plt.show()


if __name__ == '__main__':
    main()
```

Output :



```
zy-classifier-master$ pyt
 python3            python3.5-config   python3.5m-config  python3.7-config   python3.7m-config  python3m           python-config
 python3.5          python3.5m         python3.7          python3.7m         python3-config     python3m-config
zy-classifier-master$ python main.py
                                                                                              | 30/30 [00:13<00:00,  2.24it/s]
zy-classifier-master$ python main.py
                                                                                              | 30/30 [00:13<00:00,  2.22it/s]
zy-classifier-master$ python main.py
                                                                                              | 30/30 [00:07<00:00,  4.08it/s]
zy-classifier-master$ python main.py
                                                                                              | 30/30 [00:07<00:00,  4.08it/s]
```

Result Analysis :

The initial implementation of the fuzzy classifier, without the improvement from GA, manages to accurately classify 115 of the 150 datasets which equals roughly to an accuracy of around 77%.

With the improvement from GA the accuracy of the classifier is significantly improved. Up to 143 out of the 150, circa 95%, sets can be accurately classified after the algorithm has been allowed to run its course.

Conclusion :

Iris dataset is classified using Fuzzy logic and Genetic Algorithm.
Learned to implement Fuzzy Logic Systems.

Future Scope :

To improve the accuracy further a couple of approaches exist. The most straightforward one would be to look at the fuzzy rules and possibly make changes to them in order to improve the accuracy further. This could possibly be an area where future work could take place.

References :

https://towardsdatascience.com/genetic-algorithm-implementation-in-python-5ab67bb124a6

https://www.neuraldesigner.com/blog/genetic_algorithms_for_feature_selection

Implementing a Fuzzy Classifier and Improving its Accuracy using Genetic Algorithms Stig-Åke Svensson Mälardalen University M.Sc.Program in Computer Engineering and Electronics.