**ASSIGNMENT NO. 5**

**TITLE**: Solve Air conditioner Controller using MATLAB Fuzzy logic tool box.

**AIM**: Implement a Fuzzy Inference System for Air Conditioner Controling using the MATLAB Fuzzy Logic Tool box.

**SAVITRIBAI PHULE PUNE UNIVERSITY**

**A PRELIMINARY PROJECT REPORT ON**

**“Air Conditioner Controller using MATLAB Fuzzy Logic Toolbox”**

**SUBMITTED TOWARDS THE PARTIAL FULFILMENT OF THE REQUIREMENTS OF**

**BACHELOR OF ENGINEERING (BE COMPUTER ENGINEERING)**

**Academic Year: 2018-19**

**By:**

**Mehul Solanki(BECOB260)**

**Vikrant Wyawahare(BECOB271)**

**Under The Guidance of**

**Dr. Anuradha Thakare**



**DEPARTMENT OF COMPUTER ENGINEERING,**

**PIMPRI CHINCHWAD COLLEGE OF ENGINEERING**

**SECTOR 26, NIGDI, PRADHIKARAN**



PIMPRI CHINCHWAD COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

**CERTIFICATE**

This is to certify that, the project entitled

**“Air Conditioner Controller using MATLAB Fuzzy Logic Toolbox**”

is successfully carried out as a mini project successfully submitted by following students of “PCET's Pimpri Chinchwad College of Engineering, Nigdi, Pune-44**”.**

**Under the guidance of Prof. Dr. Anuradha Thakare**

In the partial fulfilment of the requirements for the B.E. (Computer Engineering)

**Mehul Solanki**

**Vikrant Wyawahare**

**Dr. Anuradha Thakare**

**Project Guide**

**Abstract**

With the exponential increase in the use of cooling device, the air conditioning systems are becoming an essential part of our day to day life. Data suggest an exponential rise in the use of air conditioners in urban as well as rural India. With the increase in the usage of air conditioners, there is a simultaneous increase in the electrical power consumption. In this mini-project a design has been proposed considering various input parameters and applying Fuzzy Logic System to the Air Conditioner. By considering the input parameters we can greatly modify the functioning of the AC and reduce the electrical energy intake of the AC compressor/Fan while utilizing all available resources in the efficient manner. MATLAB Fuzzy logic toolbox is used to build the Fuzzy Inference Air Conditioner Controller System.

**Index**

|  |  |  |  |
| --- | --- | --- | --- |
| **Chapter** | **Contents** | | **Page No.** |
| 1. | Introduction | | 1 |
|  | a. | Problem Statement | 1 |
|  | b. | Project Idea | 1 |
|  | c. | Motivation | 1 |
|  | d. | Scope | 1 |
|  | e. | Literature Survey/ Requirement Analysis | 2 |
| 2. | Project Design | | 3 |
|  | a. | Technological Stack | 3 |
|  | b. | Dataset Design | 3 |
| 3. | Module Description | | 4 |
| 4. | Results | | 6 |
|  | a. | Source code | 6 |
|  | b. | Screenshots including GUI | 12 |
|  | c. | Test cases | 13 |
| 5. | Conclusion | | 14 |
| 6. | References | | 15 |

List of Figures & Tables

|  |  |  |
| --- | --- | --- |
| **Figure/Table No.** | **Figure/ Table Name** | **Page No.** |
| 1 | System Architecture | 4 |
| 2 | Flowchart for GA | 5 |
| 3 | Screenshot: Input Form | 12 |
| 4 | Screenshot: Output (City Plot) | 12 |
| 5 | Screenshot: Output (Results) | 13 |

**Chapter 1**

**Introduction**

* 1. **Problem Statement**

This project presents us with a feature selection problem in automated design of pattern classifiers. The feature subset selection problem refers the task of identifying and selecting a useful subset of features to be used to represent patterns from a larger set of often mutually redundant, possibly irrelevant, features with different associated measurement costs and/or risks.

* 1. **Project Idea**

The project idea is to create a web application which will be interactive with the user so that the input parameters are entered by the user correctly. On submission of the input parameter form, the server will run the genetic algorithm with the specified parameters and generate an optimal solution.

* 1. **Motivation**

Given the large number of features, it is difficult to find the subset of features that is useful for a given task. Genetic Algorithms (GA) can be used to alleviate this problem, by searching the entire feature set, for those features that are not only essential but improve performance as well. In this project, we explore the various approaches to use GA to select features for different applications, and develop a solution that uses a reduced feature set (selected by GA) to classify images based on their domain/genre.

* 1. **Scope**

As an extension of this work, we can reuse the solution proposed in this project in other scenarios, such as, to differentiate computer-generated images from camera-generated images, or identify a sketch version of an image from other genres/domains. We can also focus on improving the set of features extracted from images to cater to a wide variety of classification applications. A study comparing GA-based feature selection with other wrapper methods is also planned.

* 1. **Literature Survey**

With the increase in data available at our disposal, plus tens to hundreds of features available for different datasets, the complexity of the system increases not only in terms of understanding data, but in terms of resource utilization and system performance. While the size of the dataset cannot be controlled, the feature set can be reduced to include only relevant and unique features so that the overall performance increases and resource utilization decreases [1]. Redundant or irrelevant features may be of the form of correlated features in which there is dependency between them. The dependent features may not provide any extra information or have an impact on the output. This means that eliminating such a feature does not affect the total information content. In some cases, such features may introduce a bias in the system and thus affect the performance. Given that there may be N features possible for a dataset, there may be 2N combinations of features to test to find out which features contribute positively to the outcome of the problem. Evolutionary algorithms such as Genetic Algorithms (GA), can be used for feature selection, where a subset of features must be found from a very large search space.

**Chapter 2**

**Project Design**

**2.1 Technological Stack**

To implement the project, the following technological stack is considered:

Frontend : HTML, CSS, Bootstrap, Javascript.

Backend : Python 3, Python-Flask.

Operating System : As the project is a web application, the project is OS independent.

**2.2 Dataset Design**

**Data Set Information:**

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.   
  
I think that the initial data set had around 30 variables, but for some reason I only have the 13 dimensional version. I had a list of what the 30 or so variables were, but a.) I lost it, and b.), I would not know which 13 variables are included in the set.   
  
The attributes are (dontated by Riccardo Leardi, riclea **'@'** anchem.unige.it )   
1) Alcohol   
2) Malic acid   
3) Ash   
4) Alcalinity of ash   
5) Magnesium   
6) Total phenols   
7) Flavanoids   
8) Nonflavanoid phenols   
9) Proanthocyanins   
10)Color intensity   
11)Hue   
12)OD280/OD315 of diluted wines   
13)Proline   
  
In a classification context, this is a well posed problem with "well behaved" class structures. A good data set for first testing of a new classifier, but not very challenging.

**Attribute Information:**

All attributes are continuous   
  
No statistics available, but suggest to standardise variables for certain uses (e.g. for us with classifiers which are NOT scale invariant)

**Chapter 3**

**Module Design**

**3.1 Genetic Algorithm**

**Selection :**

We implement a probabilistic binary tournament selection. Tournament selection holds *n* tournaments to choose *n* individuals. Each tournament consists of sampling 2 elements of the population and choosing the best one with a probability *p ∈* [0*.*5*,* 1].

**Mutation :**

The mutation is an operator which allows diversity. During the mutation stage, a chromosome has a probability *pmut* to mutate. If a chromosome is selected to mutate, we choose randomly a number *n* of bits to be flipped then *n* bits are chosen randomly and flipped. In order to create a large diversity, we set *pmut* around 10% and *n ∈* [1*,* 5].

**Crossover :**

We use a Subset Size-Oriented Common Feature Crossover Operator (SSOCF) which keeps useful informative blocks and produces offsprings which have the same distribution than the parents. Offsprings are kept, only if they fit better than the least good individual of the population. Features shared by the 2 parents are kept by offsprings and the non shared features are inherited by offsprings corresponding to the *ith* parent with the probability (*ni − nc/nu*) where *ni* is the number of selected features of the *ith* parent, *nc* is the number of commonly selected features across both mating partners and *nu* is the number of non-shared selected features.



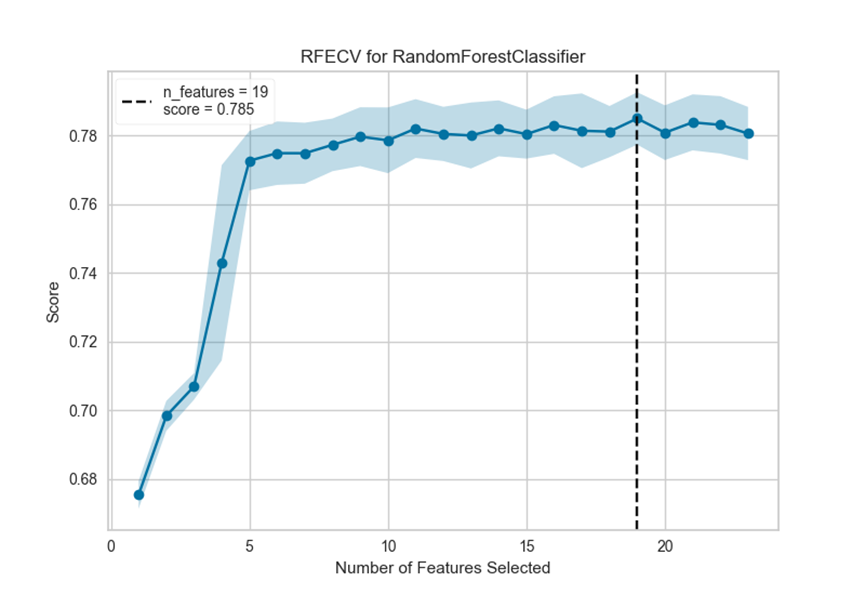
**3.2 Random Forest Feature Importance :**

# Random forests are among the most popular machine learning methods thanks to their relatively good accuracy, robustness and ease of use. They also provide two straightforward methods for feature selection: mean decrease impurity and mean decrease accuracy.

Mean decrease impurity :

Random forest consists of a number of decision trees. Every node in the decision trees is a condition on a single feature, designed to split the dataset into two so that similar response values end up in the same set. The measure based on which the (locally) optimal condition is chosen is called impurity. For classification, it is typically either Gini impurity or [information gain/entropy](http://en.wikipedia.org/wiki/Information_gain_in_decision_trees) and for regression trees it is [variance](http://en.wikipedia.org/wiki/Variance). Thus when training a tree, it can be computed how much each feature decreases the weighted impurity in a tree. For a forest, the impurity decrease from each feature can be averaged and the features are ranked according to this measure.

## Mean decrease accuracy :

Another popular feature selection method is to directly measure the impact of each feature on accuracy of the model. The general idea is to permute the values of each feature and measure how much the permutation decreases the accuracy of the model. Clearly, for unimportant variables, the permutation should have little to no effect on model accuracy, while permuting important variables should significantly decrease it. This method is not directly exposed in sklearn, but it is straightforward to implementit.

**3.3 Recursive Feature Elimination :**

Recursive feature elimination (RFE) is a feature selection method that fits a model and removes the weakest feature (or features) until the specified number of features is reached. Features are ranked by the model’s attributes, and by recursively eliminating a small number of features per loop, RFE attempts to eliminate dependencies and collinearity that may exist in the model.

RFE requires a specified number of features to keep, however it is often not known in advance how many features are valid. To find the optimal number of features cross-validation is used with RFE to score different feature subsets and select the best scoring collection of features. The RFECVvisualizer plots the number of features in the model along with their cross-validated test score and variability and visualizes the selected number of features.

**Chapter 4**

**Results**

**4.1 Codes :**

# Data

dataset = load\_wine()

X, y = dataset.data, dataset.target

features = dataset.feature\_names

# CV MSE before feature selection

est = LinearRegression()

score = -1.0 \* cross\_val\_score(est, X, y, cv=5, scoring="neg\_mean\_squared\_error")

print("CV MSE before feature selection: {:.2f}".format(np.mean(score)))

# CV MSE after feature selection: RFE

rfe = RFECV(est, cv=5, scoring="neg\_mean\_squared\_error")

rfe.fit(X, y)

score = -1.0 \* cross\_val\_score(est, X[:,rfe.support\_], y, cv=5, scoring="neg\_mean\_squared\_error")

print("CV MSE after RFE feature selection: {:.2f}".format(np.mean(score)))

# CV MSE after feature selection: Feature Importance

rf = RandomForestRegressor(n\_estimators=500, random\_state=SEED)

rf.fit(X, y)

support = rf.feature\_importances\_ > 0.01

score = -1.0 \* cross\_val\_score(est, X[:,support], y, cv=5, scoring="neg\_mean\_squared\_error")

print("CV MSE after Feature Importance feature selection: {:.2f}".format(np.mean(score)))

# Class performing feature selection with genetic algorithm

class GeneticSelector():

def \_\_init\_\_(self, estimator, n\_gen, size, n\_best, n\_rand,

n\_children, mutation\_rate):

# Estimator

self.estimator = estimator

# Number of generations

self.n\_gen = n\_gen

# Number of chromosomes in population

self.size = size

# Number of best chromosomes to select

self.n\_best = n\_best

# Number of random chromosomes to select

self.n\_rand = n\_rand

# Number of children created during crossover

self.n\_children = n\_children

# Probablity of chromosome mutation

self.mutation\_rate = mutation\_rate

if int((self.n\_best + self.n\_rand) / 2) \* self.n\_children != self.size:

raise ValueError("The population size is not stable.")

def initilize(self):

population = []

for i in range(self.size):

chromosome = np.ones(self.n\_features, dtype=np.bool)

mask = np.random.rand(len(chromosome)) < 0.3

chromosome[mask] = False

population.append(chromosome)

return population

def fitness(self, population):

X, y = self.dataset

scores = []

for chromosome in population:

score = -1.0 \* np.mean(cross\_val\_score(self.estimator, X[:, chromosome], y,

cv=5,

scoring="neg\_mean\_squared\_error"))

scores.append(score)

scores, population = np.array(scores), np.array(population)

inds = np.argsort(scores)

return list(scores[inds]), list(population[inds, :])

def select(self, population\_sorted):

population\_next = []

for i in range(self.n\_best):

population\_next.append(population\_sorted[i])

for i in range(self.n\_rand):

population\_next.append(random.choice(population\_sorted))

random.shuffle(population\_next)

return population\_next

def crossover(self, population):

population\_next = []

for i in range(int(len(population) / 2)):

for j in range(self.n\_children):

chromosome1, chromosome2 = population[i], population[len(population) - 1 - i]

child = chromosome1

mask = np.random.rand(len(child)) > 0.5

child[mask] = chromosome2[mask]

population\_next.append(child)

return population\_next

def mutate(self, population):

population\_next = []

for i in range(len(population)):

chromosome = population[i]

if random.random() < self.mutation\_rate:

mask = np.random.rand(len(chromosome)) < 0.05

chromosome[mask] = False

population\_next.append(chromosome)

return population\_next

def generate(self, population):

# Selection, crossover and mutation

scores\_sorted, population\_sorted = self.fitness(population)

population = self.select(population\_sorted)

population = self.crossover(population)

population = self.mutate(population)

# History

self.chromosomes\_best.append(population\_sorted[0])

self.scores\_best.append(scores\_sorted[0])

self.scores\_avg.append(np.mean(scores\_sorted))

return population

def fit(self, X, y):

self.chromosomes\_best = []

self.scores\_best, self.scores\_avg = [], []

self.dataset = X, y

self.n\_features = X.shape[1]

population = self.initilize()

for i in range(self.n\_gen):

population = self.generate(population)

return self

@property

def support\_(self):

return self.chromosomes\_best[-1]

def plot\_scores(self):

plt.plot(self.scores\_best, label='Best')

plt.plot(self.scores\_avg, label='Average')

plt.legend()

plt.ylabel('Scores')

plt.xlabel('Generation')

plt.show()

sel = GeneticSelector(estimator=LinearRegression(),

n\_gen=7, size=200, n\_best=40, n\_rand=40,

n\_children=5, mutation\_rate=0.05)

sel.fit(X, y)

sel.plot\_scores()

score = -1.0 \* cross\_val\_score(est, X[:, sel.support\_], y, cv=5, scoring="neg\_mean\_squared\_error")

print("CV MSE after GA feature selection: {:.2f}".format(np.mean(score)))

**4.2 Screenshots**

**4.3 Test Cases**

**Chapter 5**

**Conclusion**

To demonstrate that feature selection not only improves execution time, but also improves

classification accuracy, we utilized a GA to select feature subsets from the entire set of features, considering the classifier accuracy as the fitness function. Utilizing this reduced feature set for image classification, the results showed that feature selection improves the accuracy of classification in the case of SVM and Neural Networks, while not making an impact on the classification accuracy in the case of Random Forests. This was expected, as the performance of Random Forests is known to be unaffected by the features selected.

From our results, we can conclude that GA has a positive impact on the performance of

classification, and SVM along with feature selection, performs the best for the classification

considered.

**Chapter 6**

**References**

[1] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," Computers and

Electrical Engineering, vol. 40, no. 1, pp. 16-28, 2014.

[2] C. Bates Congdon. A comparison of genetic algorithm and other machine learning systems on a complex classification task from common disease research. PhD thesis, University of Michigan, 1995.

[3] C. Emmanouilidis, A. Hunter, and J. MacIntyre. A multiobjective evolutionary setting for feature selection and a commonality-based crossover operator. In Congress on Evolutionary Computing 2000, volume 2, pages 309–316. CEC, 2000.

[4] J. Horn, D.E. Goldberg, and K. Deb. Implicit niching in a learning classifier system : Nature’s way. Evolutionary Computation, 2(1):37–66, 1994.

[5] S.W. Mahfoud. Niching Methods for Genetic Algorithms. PhD thesis, University of Illinois, 1995.

[6] N. Monmarch´e, M. Slimane, and G. Venturini. Antclass : discovery of cluster in numeric data by an hybridization of an ant colony with the kmeans algorithm. Technical Report 213, Ecole d’Ing´enieurs en Informatique pour l’Industrie (E3i), Universit´e de Tours, Jan. 1999.

[7] M. Pei, E.D. Goodman, and W.F. Punch. Feature extraction using genetic algorithms. Technical report, Michigan State University : GARAGe, June 1997.