Extreme ANFIS

abhinav.tushar.vs@gmail.com

The overall aim is to make an adaptable sugeno fuzzy inference system with **n** inputs and **1** output with the adaptable part handled by ELM.

Why this document ?

Becuase I was getting heavily confused in exact matrix multiplications to do and to implement in code. This is a helper doc for that.

Matrix notes

- Matrix representation **m x n** represents a matrix with **m** rows and **n** columns.
- **[1, 2, 3]** creates a **3 x 1** matrix (also a vector)
- **[1 2 3]** creates a **1 x 3** matrix
- **[1 2 3; 4 5 6]** creates a **2 x 3** matrix
- **tmp_** is appended to temporary variables, all normal variables are accessible globally in the doc

Variables

x_train : **n x n_vars** represents the training input data.

n is the number of observations **n_vars** is the number of variables in input

y : **n x 1** represents the output

n is the number of observations

Implementation

In regular Neural networks, the choice of number of hidden nodes is on the user. ANFIS is no different. Here, for every input variable, the user has to specify number of membership functions.

In Extreme ANFIS, these membership functions are selected according to few conditions [1]. Even if these functions are selected as bell functions and distributed uniformly over the range of each input, those conditions are essentially satisfied.

The steps are :

- **Create the membership functions for each input variables**
  - Say you have **n_vars** number of variables, and **n_mf** number of membership functions for each.

    - Then at next layer, you have **n_vars** **n_mf* things that will have different values in the range of 0 to 1 (since all those are membership functions)
- **Now comes the number of rules**
    - The number of rules **n_rules** governs the number of output equations that are to be summed by weighing.
    - Assuming every possible combination of input membership functions gives us the maximum (and most adaptable and smooth) output.
    - This leads to **n_rules = n_mf ^ n_vars**
    - These rules also have a normalised weight matrix **wt** : **n_rules x 1**
    - All these rules are linear equations having **n_vars + 1** coefficients each. This leads to total parameters on consequence side = **(n_vars + 1) * n_rules**
    - For every rule, there is a firing strength (the weight)
    - The system till now should be able to do the following

      ```
      tmp_single_weights = {the system}(tmp_single_observation)
      ```

      Here, **tmp_single_weights** is an **n_rules x 1** matrix, and **tmp_single_observation** is an **n_vars x 1** matrix

Following this, the system will be made to handle multiple inputs

```
weights = {the system}(observations)
```

Here, **weights** is an **n x n_rules** matrix and **tmp_observations** is an **n x n_vars** matrix

- **Now the ELM thing**

For **n** observations **x** : **n x n_vars**, the output **y** : **n x 1** can be found by

```
weights = {the system}(x)
# `weights` is `n x n_rules` matrix

# `parameters` is the matrix of parameters to tune
# Its shape is `1 x ((n_vars + 1) * n_rules)`

y = parameters * H
```

Now, parameters can be found using

```
parameters = y * pinv(H)
```

Here, **H** is a matrix that has shape **n_rules * (n_vars + 1) x n**. In **H**, each column represents an observation and is like this . . .

```
[w1 * var1, w1 * var2, ... w1 * varn_vars, w1, w2 * var1, w2 * var2, . . . .]
```

Here, **wi** is the weight of rule **i** for that observation, while **vari** is the variable **i** value. This **H** matrix can be generated as

```
# `x` : `n x n_vars`
```

`weights` : `n x n_rules`

cbind(x, [1, 1, 1. . . n times]) # Appending a column

now `x` is `n x (n_vars + 1)`

x = x'

now `x` is `(n_vars + 1) x n`
multiplying each element of a row of `weights` to column of x
will generate a `new_x` for each column of `weights`

# binding each `new_x` using row
H = rbind(`new_x1`, `new_x2` . . . .)

- **Prediction**

  y = parameters * H

References

[1] Pushpak Jagtap, G.N. Pillai, "Comparison of Extreme-ANFIS and ANFIS Networks for Regression Problems", 2014 IEEE International Advance Computing Conference (IACC)