CS 550 Machine Learning
Homework 3

Muhammed Çavuşoğlu, 21400653

December 18, 2018

In this homework, I have designed and implemented a genetic algorithm based approach for cost-sensitive multiclass classification. My implementation details are explained in the following section, and results that I obtained on the Thyroid dataset are explained in Results section.

Implementation

My implementation details are explained in the following subsections.

Representation

I have used a bit array representation to indicate which features are selected. For example, in this array [0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1], there are 21 elements for 21 features, and features represented with 1 are selected and features represented with 0 are not selected. In my implementation, I refer to each bit array as an *individual*, and groups of individuals as a *population*. When creating an *individual*, I randomly select 0 and 1 values. Since the 21^{st} feature is a combination of 19^{th} and 20^{th} features; when creating an individual randomly, if 21^{st} feature is selected, I select 19^{th} and 20^{th} features as well.

Classification Algorithm & Fitness Function

I have used *sklearn*'s decision tree classifier [1] for multiclass classification due to its simplicity. My fitness function takes an individual (i.e. [0, 1, ..., 1]), and calculates a fitness value based on the misclassification cost and the cost of extracting the selected features. It selects features of the dataset according to that individual, builds a decision tree classifier for that training set, and calculates accuracy values.

Initially, I set my fitness value to be the multiplication of the mean accuracy and feature selection cost to take both costs into account. However, I observed that in some cases, even though the mean accuracy is around 99%, first and second class accuracies are lower. In order to prevent this, I updated my fitness value calculation to take class accuracies into account. As a result, my fitness value is equal to (misclassification percentage of class 1 $*$ misclassification percentage of class 2 $*$ misclassification percentage of class 3 $*$ feature selection cost). For example, if class 1 accuracy is 80%, class 2 accuracy is 90%, class 3 accuracy is 95%, and feature selection cost is 40; fitness value becomes $20*10*5*40 = 40000$. Lower fitness value indicates a fitter individual. Pseudocode of my fitness function is given below.

Algorithm 1 Fitness function

```
function FITNESS(individual)                ▷ determines the fitness of an individual
    fs_cost ← calculate feature selection cost
    filter dataset based on selected features
    build a decision tree classifier for that training set
    predict the test set, and calculate misclassification percentages (mp) for each class

    return (mp of class 1 ∗ mp of class 2 ∗ mp of class 3 ∗ fs_cost)
```

Evolution

Genetic algorithm based operations take place in *evolve* function. My main function initializes a population and uses evolve function to improve the population. Pseudocode for this main function is given below.

Algorithm 2 Main function

```
pop ← initialize a population with a specified size
fitness_history ← empty set

for max number of iterations do                ▷ it usually converges before this
    pop ← evolve(pop)
    pop_fitness ← average fitness of population
    append pop_fitness to fitness_history

    if pop_fitness < 250 then                  ▷ convergence
        break
```

In the pseudocode above, the convergence check 250 is based on my experiments. 250 for the average population fitness value turned out to be very low, and the individuals in that population tended to be the fittest ones with 95%+ class accuracies, 99%+ mean accuracies, and feature selection cost of around 54. I tested my implementation with a population of size 20, and maximum number of iterations of 100 (it usually converged before 100 iterations).

The *evolve* function used in my main function improves the population. Its parameters are retain percentage, random selection probability, and mutation probability. Retain percentage controls the number of individuals to select as parents from the previous generation. Random selection probability controls the random selection of non-parent individuals process to increase diversity. Mutation probability controls the chance of mutation. I fine-tuned these parameters and the following values yielded fit individuals: retain percentage = 0.50, random selection probability = 0.05, and mutation probability = 0.01.

The pseudocode for the *evolve* function is given below.

Algorithm 3 Evolve function

```
function EVOLVE(population, retain percentage, random selection prob, mutation prob)
    f_values ← array of fitness values for each individual in the population
    individuals ← array individual values (strings) in the order of increasing fitness values
    retain_length ← size of the population ∗ retain percentage
    parents ← first retain_length members of individuals

    for each member of individuals that are not in parents do        ▷ increases diversity
        if random selection prob > random() then
            append that member to parents

    for each member of parents do                                     ▷ mutate
        if mutation prob > random() then
            randomly find the index to mutate
            randomly set it to 0 or 1

            if 21st feature is selected then                          ▷ make sure the result is valid
                select 19th and 20th features as well

    children ← empty set                                              ▷ crossover
    remaining no of individuals ← population size - size of parents

    while size of children < remaining no of individuals do
        male_index ← randomly select an index from parents
        female_index ← randomly select an index from parents

        if male_index ≠ female_index then
            child ← append the first half of male individual and the second half of female individual
            append child to children

    append children to parents
    return parents
```

Results

As a result of my algorithm, the fittest individual is [0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1]. In other words; 3^{rd}, 7^{th}, 8^{th}, 12^{th}, 16^{th}, 17^{th}, 19^{th}, 20^{th}, 21^{st} features are selected. Total cost of these selected features is 53.7.

Using the features selected by the fittest individual, my classifier could classify all 3772 instances in the training set correctly, so the training accuracy for each class is 100%, and the mean training accuracy is also 100%. Test set accuracies are given in the following table.

Class	Accuracy
Class 1	95.89%
Class 2	100.00%
Class 3	99.31%
Mean accuracy: 99.27%	

Table 1: Test accuracies of the fittest individual

References

[1] "sklearn.tree.DecisionTreeClassifier." `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`. [Accessed: December 12, 2018].