CS550 Machine Learning Homework 3 Report

Proposed to: Cigdem Gunduz Demir

Emre Doğan
Department of Computer Engineering
Bilkent University
Ankara, Turkey
emre.dogan@bilkent.edu.tr

1 INTRODUCTION

This report is written to illustrate the technical details of genetic algorithm-based approach for cost sensitive learning, in which the misclassification cost is considered together with the cost of feature extraction.

The learning model is applied on the Thyroid Dataset[1] which has a unbalanced sample distribution. In its webpage, it is stated that "Because 92 percent of the patients are not hyperthyroid a good classifier must be significant better than 92%."[1] So, my main purpose on this study is to achieve a accuracy better than 92% and not a biased one.

2 ALGORITHM

The implemented algorithm consists of 3 main parts:

2.1 Data Preprocessing:

In this study, "Thyroid data set" from UCI repository with its own separate training and test datasets is used. When the class label distribution is examined, it was observed that 92% of training set belongs to a single output label. Regardless of the chosen classifier, this distribution tends to learn a bias through the majority label. To be able deal with this issue, I used oversampling method to create a more balanced dataset. The samples with minority labels are duplicated 20 times in both training and test dataset ending up with a training set with 9452 samples and test set with 8428 samples.

2.2 Classifier:

As classifier, I tried several algorithms such as: Naïve Bayes classifier, Support Vector Machine, Neural Network etc. At the end, I decide upon working with neural networks by using *scikit-learn*[1], a machine learning library for Python language. The preferred neural network architecture is a one hidden layer neural network with 100 nodes within it.

2.3 Genetic Algorithm:

To be able to implement the genetic algorithm, the features of initial members ,10 for our case, to be used in classification are encoded in bit string format. As there are 21 features in total, member representations are in 21 bits consisting of 0s and 1s. '1' represents that this feature will be used in the classification and '0' shows that the classifier will not be affected by this feature.

A sample bit encoded hypothesis can be observed on Figure 1.

```
[0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0]
```

Figure 1. A sample hypothesis that is encoded in bit-string format.

The hypothesis in Figure 1 consists of 21 bits representing features. For the evaluation of this hypothesis, only 5^{th}, 7^{th}, 13^{th}, 16^{th} and 20^{th} features will be considered in the classification and the rest of features will not be taken into account.

After having all initial members of population with their bit-string representations, the neural network classifier is implemented for each member of population. Then, corresponding accuracy, F1 Score and feature cost values are kept in some lists. These lists are length of population size(10 in my implementation) all the time. Some new members are added and mutated, but the total number of members ,so the length of these lists, in the population never changes.

These lists are:

- *indicesListPopulation:* keeps the bit string representation of each member hypothesis.

- *howManyFeatures:* lists the number of selected features.

- *accuracyListPopulation:* keeps the accuracies of members.

- *costListPopulation:* keeps the feature cost for each member.

- *fitnessPopulation:* keeps the fitness value for each member.

The fitness function is defined by considering both misclassification error and feature extraction cost. To construct the fitness function, I used decomposition method. Each objective value is applied min-max normalization and a weight vector is given for each objective. My objectives are to maximize accuracy and minimize feature cost. I also used F1 score to detect if there exists a bias in the model (92% of the original dataset belongs to the label 3). If F1 Score is smaller than 0.7, then the calculated weighted fitness function is multiplied by 0.5 as a penalty.

The pseudo-code of the fitness function is given below:

```
FITNESS_FUNCTION(indices):
            w1 = 0.5
            w2 = 0.5
    if (f1Val ≥ 0.7):
        fitness = w1 ∗ Acc + w2 ∗ (1 − FeatureCost/102.34)
    else:
        fitness = 0.5 ∗ (w1 ∗ Acc + w2 ∗ (1 − FeatureCost/102.34))
    return fitness
```

After initializing the population, the repeating evolutionary process begins. At each iteration of this loop, crossover and mutation operation take place. For crossover, 2 random members are chosen and applied single point crossover. From the resulting offspring, two new members are created and appended to the population. Also, all the lists of evaluation metric (accuracy, cost, fitness) are updated. Just after this addition of 2 new members, 2 hypotheses with the lowest fitness values are killed to protect the number of members in the population.

After the crossover, the mutation is applied to a ratio of the whole population. A variable 'ratioToMutate' is defined such that (ratioToMutate x sizeOfPopulation) members are mutated at a single point.

At the end of each iteration, the highest and mean fitness values are kept in separate lists so that we can observe the improvement in the fitness values.

The genetic algorithm is implemented in the main script. The pseudo code of it can be seen below.

```
GENETIC_ALGORITHM(F1Val, Acc, FeatureCost):
    Initialize the population with 10 members.
    for each member of population:
        f1S, Acc ← classify(X_tr,X_te,y_tr,y_te)
    numOfCouplesToCrossover = 2
    ratioToMutate = 0.2
    for i ← 1 to numOfIterations:
        mem1,mem2 ←Choose randomly 2 members.
        offs1,offs2 ← crossover(mem1,mem2)
        Feed offs1& offs2 to Classifier.
        listOfIndices.append(offs1,offs2)
        Update evaluation metrics of offsprings.
        for ratioToMutatexsizeOfPopulation elmnts:
            mutation(randomChoose(member))
            Feed member to Classifier.
            Update evaluation metrics of member.
        delete 2 members with min. fitness values.
```

The crossover and mutation functions used in genetic algorithm are defined separately and given in pseudocode below.

```
CROSSOVER(member1,member2):
    pnt ← randomint(len(member1))
    offs1 ← member1[:pnt] + member2[pnt:]
    offs2 ← member1[pnt:] + member2[:pnt]
    return offs1,offs2


MUTATION(indices):
    pnt ← randomint(len(indices))
    if (indices[pnt]==0):
        indices[pnt]=1
    else if (indices[pnt]==1):
        indices[pnt]=0
```

3 EVALUATION

As mentioned in Part 2, the fitness function is defined as,

fitness = w1 ∗ Acc + w2 ∗ (1 − FeatureCost/102.34) if F1Score≥0.7

In unbalanced data distribution, the classifier may lead to a bias towards the majority label. When I first examined the distribution of Thyroid Dataset, I observed that most of samples belong to label 3 and the accuracy did not show a meaningful result.(Although it was 92%, class accuracies were not balanced.) To solve this problem, I used 2 solutions,

1. Oversampling the minority labels before the process.
2. Using F1 Score as a decision metric in fitness function.

$$F1Score = 2 * \frac{precision * recall}{precision + recall}$$

The definition of F1 Score is given above. It is a more meaningful metric when dealing with imbalanced distribution. So, if the F1 Score is less than 0.7, then the fitness value is decreased by 2 as a penalty. By doing this, I ensured that no hypothesis that suffers from imbalanced data issue will be able to survive in the population.

Another issue on the fitness function is the choice of w1 and w2 weights. I tried different combinations of weights to observe the difference and ended up with choosing w1=w2=0.5. If the algorithm is required to be more accuracy sensitive, then w1 should be increased (w1=0.7, w2=0.3). If it is required to be more cost sensitive, vice versa.

The stopping criteria of genetic algorithm was not stated explicitly. Instead, I started the genetic algorithm with very large

number of iterations and observed that after some point, the fitness function cannot enhance itself and reaches to the saturation level. This happened around the iteration 200. So, I examined all the evaluation part with 250 iterations.

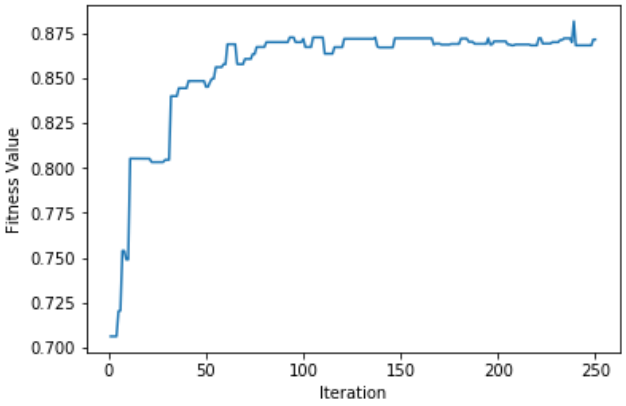

Figure 2. Best Fitness Values for 250 iterations.

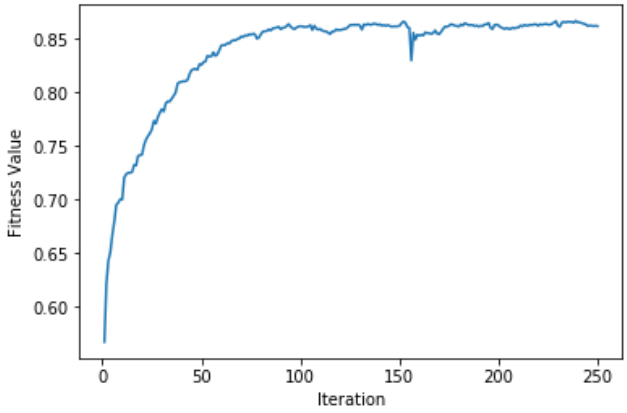

Figure 3. Average Fitness Values for 250 iterations.

To evaluate the genetic algorithm and cost sensitive learning success, I stored the best and average fitness values of hypotheses for all iterations. The plots for best and average fitness values can be seen from Figure 2 and 3. My fitness function definition is in the range 0 and 1. As it can be observed, the fitness value starts with the value around 0.60 and increases through 0.85.

4 RESULTS

After 250 iteration of genetic algorithm, I ended up with the hypothesis with following features.

Bit-String Encoding	[0,0,1,1,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0]
Selected Features	on_thyroxine[3], query_on_thyroxine[4], goitre[13], TSH[17]
Final Accuracy	0.954
Total Feature Cost	25.780
Final Fitness	0.871

Table 1. Features of the Final Hypothesis of Genetic Algorithm.

After the desired features are achieved, the training and testing accuracy results are calculated. They can be observed from Table 2.

Note that these values are achieved with oversampled training data.

With All 21 Features		With 4 Selected Features	
Training Acc.	Test Acc.	Training Acc.	Test Acc.
0.973	0.975	0.951	0.969

Table 2.Accuracy Values with All Features and Selected Features.

5 CONCLUSIONS

In this homework, a genetic algorithm-based cost sensitive learning model is implemented. For genetic algorithm, I defined a fitness function based on both misclassification error and feature extraction cost. This fitness function can be flexed if a more cost sensitive or accuracy sensitive system is desired.

As given in Results section, by using only 4 of 21 features, a very accurate model can be implemented. The results illustrate that a greater number of features does not guarantee a more successful model. We could achieve very similar results with almost 1/3 of the total feature cost.

As a result, cost sensitive learning is a very useful and profitable approach for machine learning applications. If it is used properly, it can save a considerable amount of time and resource.

REFERENCES

[1] Archive.ics.uci.edu. (2018). [online] Available at: https://archive.ics.uci.edu/ml/machine-learning-databases/thyroid-disease/ann-thyroid.names [Accessed 17 Dec. 2018].

[2] Scikit-learn.org. (2018). scikit-learn: machine learning in Python — scikit-learn 0.20.1 documentation. [online] Available at: https://scikit-learn.org/stable/ [Accessed 17 Dec. 2018].

[3] N. Thai-Nghe, Z. Gantner and L. Schmidt-Thieme, "A new evaluation measure for learning from imbalanced data," The 2011 International Joint Conference on Neural Networks, San Jose, CA, 2011, pp. 537-542. doi: 10.1109/IJCNN.2011.6033267

[4] Cs.bilkent.edu.tr. (2018). CS 550: Machine Learning. [online] Available at: http://www.cs.bilkent.edu.tr/~gunduz/teaching/cs550/ [Accessed 17 Dec. 2018].