Neural Net Feature Selection using a Genetic Algorithm

Katie Abrahams
CS406 (Project: Machine Learning)
Winter Term, 2016

Introduction

Selecting a subset of features to use in neural network training has the potential to improve a number of classifier characteristics. By using analysis of classifier features to optimize feature selection, classifier performance can be improved. Knowing which features are most relevant to classifier training requires analysis of accuracies using a selection of features to reveal which have the biggest impact on overall classifier accuracy. Using a genetic algorithm, one can discover which features are most helpful in training a classifier to correctly classify data. The characteristics examined in Yang and Honavar's paper [4] include accuracy, speed of classification, number of training examples needed, and classification performance cost. The adaptation of Yang's project for this paper will focus on neural network accuracy with feature selection, using the UCI Letter Recognition Data Set [2].

Specific Questions of this Project

The motivation of Yang and Honavar was to improve classifier accuracy and performance by using a subset of features. Specifically, Yang and Honavar looked at "an approach to the multi criteria optimization problem of feature subset selection" [4, p.1] To do so, Yang and Honavar focused on four attributes of a classifier than can be affected by the choice of features: classification accuracy, time to gain a sufficiently accurate classification function, number of examples needed for sufficient accuracy, and the cost of

classification. Cost in this context can be monetary or otherwise, such as risk of a surgical procedure. [4]

The wide-ranging applications of improving classifier performance appealed to Yang and Honavar. Potential applications mentioned in the paper [4] included medical diagnoses, focused on cost or risk analysis. Selecting a subset of clinical tests with differing cost, diagnostic value, and risk to be performed on a patient was mentioned as being of significant interest. In more general terms, any task that required "practical pattern classification and knowledge discovery" [4, p.1] could be improved by improving classifier speed and accuracy. If substantial improvements could be made to classifiers, then they could be more widely deployed at a reduced cost.

Solving these questions of improving pattern classification is important to machine learning because classification applies to a wide range of problems, both theoretical (in terms of optimization of a widely-used classifier type), and practical. Medical applications featured prominently in Yang and Honavar's paper, but the applications are not limited to medicine. Any tasks that require pattern classification can be aided by improving the classifiers used to recognize those patterns.

Prior Work

Yang and Honavar used a genetic algorithm to select a subset of features to train a neural net. Instead of using stochastic gradient descent (running the risk of encountering local minima), they chose to use a constructive neural network learning algorithm, DistAI. Use of DistAI guaranteed convergence for classification accuracy on any finite training set. [4, p.6-7].

Yang and Honavar used a standard genetic algorithm with rank based selection [4, p. 7]. Feature selection was done with a binary vector of size *m* (with *m* total attributes). A '1' in the vector string at position *i* meant that feature *i* would be used in neural net training. A '0' meant that feature would be left out.

For their experiments, Yang and Honavar used 10 different training/test sets, from the machine learning repository at the University of California at Irvine. They implemented 10-fold cross-validation to get their results, which they averaged to get overall results [4, p.10-11]. Their fitness function appears in Figure 1, taken from [4, p.8].

$$fitness(x) = accuracy(x) - \frac{cost(x)}{accuracy(x) + 1} + cost_{max}$$

Figure 1: Yang and Honavar's fitness function (fitness(x) is the fitness of the feature subset; accuracy(x) is the accuracy of the classifier trained with DistAI)

Their results can be found in Figure 2, taken from [4, p. 11]. Regarding accuracy-only comparison, "networks constructed using GA-selected subset of attributes compare quite favorably with networks that use all of the attributes." [4, p.10]. The combined fitness function of accuracy and cost "outperformed that of accuracy only in every aspect: the dimension, generalization accuracy and network size." [4, p.11].

	Accuracy only			Accuracy & Cost			
Dataset	*Dimension*	*Accuracy*	*Hidden*	*Dimension*	*Accuracy*	*Cost*	*Hidden*
3P	6.6 ± 1.6	100 ± 0.0	9.2 ± 4.9	4.3 ± 1.2	100 ± 0.0	26.7 ± 7.6	7.3 ± 4.2
Hepatitis	9.2 ± 2.3	97.1 ± 4.3	8.1 ± 2.8	8.3 ± 2.4	97.3 ± 3.5	19.0 ± 8.1	7.4 ± 2.8
HeartCle	7.3 ± 1.7	92.9 ± 3.6	7.6 ± 4.2	6.1 ± 1.6	93.0 ± 3.4	261.5 ± 94.4	7.2 ± 5.1
Pima	3.8 ± 1.5	79.5 ± 3.1	20.8 ± 21.2	3.1 ± 1.0	79.5 ± 3.0	22.8 ± 9.7	16.0 ± 11.1

Figure 2: Results from Yang and Honavar's feature subset selection

Yang and Honavar's results showed that feature subsets perform favorably, and that genetic algorithms are a promising way to select those feature subsets. This means that applying these methods to pattern classifiers is a promising avenue in solving practical problems.

Extension of Prior Work

This project aims to improve accuracy of a neural net by using a subset of features, drawn from feature vectors used as input to train a neural network. Yang and Honavar, the authors of the referenced paper [4], analyzed multiple classifier attributes; this project focuses on analyzing classifier accuracy across multiple epochs. Improved speed with a lessened number of features is an added bonus; timing metrics were taken for comparison, but were not the main focus of this project.

An existing neural net, from machine learning assignment 2, was modified for use with genetic algorithms. The neural net was built to classify handwritten letters, using a single layer of hidden weights. For comparison between genetic-algorithm selected feature subsets and the complete set of features, two tests were run in sequence. The first used the full feature vector to train and test the neural net, storing accuracies for comparison with the genetic algorithm modified neural net input. The second test trained the neural network using the same methods, but used only a subset of the feature vector.

In both tests, the architecture of the neural net and its methods were as follows: the numerical features representing handwritten letters [2] were used as features in the neural net forward propagation. Feature vectors representing a letter were fed as input to the neural net (with a mini-batch of size one), using forward propagation to get output. Output from the neural net represented letter classification; back-propagation with stochastic

gradient descent was used to update the weights and run the neural net again with the new weights. The use of back-propagation is in contrast to Yang and Honavar's work [4]; it was selected here to simplify the experiments (rather than add a constructive neural network learning algorithm). Epochs continued for a parameter-set number of times or until accuracy was 100%. An epoch is defined here as running input through the neural net with forward propagation to get error, and using back propagation with a momentum term to update weights between nodes. Average accuracies for each epoch were recorded, and a grand mean of average accuracies across multiple runs of epochs was used as the metric of overall classification accuracy. The choice to use multiple rounds of epochs was made in order to get averages over multiple runs of epochs; it also provided a speed boost versus many epochs on their own.

The feature subset selection was done using a string of 1s and 0s of length 17, corresponding to the 16 features of the letters and the bias input. When running forward propagation, the elements in the feature list corresponding to a '1' in the genetic algorithm string were used as features in neural net training. Features in an element matching a '0' in the genetic string were disregarded. The last element in the genetic string was always 1, to guarantee the bias was always used in training. An example from one program run appears as follows:

Genetic string ("population"):

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

Original feature vector (features corresponding to a '1' in the genetic string appear in red):

```
[-0.00419272, -0.60197147, -0.05694059, -0.59518892, 0.22847142,
0.05928873, -0.22073501, 0.50279975, 0.34053539, -0.52001135,
-0.18542856, -0.44941438, -0.01528712, -0.21902538, -0.26699267,
-1.11337607, 1.0]
```

Genetic-algorithm selected feature vector:

```
[-0.05694059, -0.59518892, -0.22073501, 0.50279975, 0.34053539,
-0.44941438, -1.113376067, 1.0]
```

The genetic algorithm used to modify the genetic population appears in this project in two forms: one, using the DEAP genetic algorithm library [1], and the other one written by this author. This author's version, simpler than the functionality provided by DEAP, was done for comparison with DEAP and for curiosity's sake. Both versions modified the population string to select different features from the feature vector (and less or more features, depending on iteration); both utilize the DEAP initial population creation `toolbox.population`, called in `initial_ga_population`.

This author's version used two functions to modify the genetic population, `genetic_cross` and `mutate`. `Genetic_cross` divided the population into two parent strings and recombined their elements to create a child population; `mutate` selected an element from the string at random if a mutation threshold was met (percentage of mutation greater than random, as defined by Python's `random` function). If the mutation threshold was met, the selected element was replaced with a 1 or a 0, potentially changing its value.

The DEAP genetic algorithm uses their library functions (seen in `gen_algorithm`, using their `toolbox` for setup) to create and evolve genetic populations. As of this writing, their final mutated strings end up in a form unusable by this project. Elements of the DEAP-trained population are no longer 1s and 0s after evolution, and are incompatible with the current method of feature subset selection.

Results

Results here correspond to the classifier's performance using this author's genetic algorithm functions.

The first surprising finding was that the initial feature subset improved average classification accuracy as compared with the genetic algorithm-trained feature selection string. Before any genetic algorithm training, the initial feature subset had ~10% better accuracies across rounds of epochs. A typical run included ~24% accuracy for the initial feature subset selection. A naïve genetic algorithm run for 10 rounds of 10 epochs (the initial benchmark) had ~14% accuracy. Using this author's mutation function, setting the threshold for mutation at 50% or higher had an adverse affect on accuracy. For comparison, a mutation threshold of 20% (mutating less often) had less of a negative effect on accuracy, returning ~23% accuracy, versus the ~14% accuracy of a 50% mutation threshold. This was unexpected; it had been assumed more mutations would yield higher fitness.

Using all features still yields improved accuracy, though runtime is slightly slower (a matter of seconds at most; this increases with more hidden units). Training accuracy using all features was ~40% for a single round of epochs (this differs from the final results seen in Table 1 and Table 2, as the tables record values for code after a refactor was done).

At this point this author realized a substantial refactor was in order. The genetic algorithm functions were re-designed to resemble the DEAP functions [1]. Mutation for multiple generations was added, as well as splitting one population string (versus the previous method of using two population strings for `genetic_cross`. `Mutate` was modified to have multiple points of mutation in the population string; previously only one element in the population could mutate per function call. Rerunning the neural net training after these changes yielded similar results for the entire feature vector and the genetic algorithm selected substring; ~25% training accuracy for both. In some test runs the neural net using entire feature vector performed better; in others the neural net with a feature subset was more accurate. Differences were typically in the range of +/- 2–3%, with ~20% accuracy.

The tests on initial feature subset selection string were used to compare with the performance of the modified feature selection string. At this point the tests for the "initial feature subset" (without genetic algorithm modification) ceased being run, as they became less useful as a comparison metric once the genetic algorithm demonstrated that it did have an effect on accuracies.

Modifications for the genetic algorithm functions written by this author brought the algorithms closer to those in the DEAP library; the choice to continue using original functions was for curiosity and education's sake. The fitness and evaluation function in particular closely followed the DEAP tutorials (choosing a sum function for evaluation). The form of the evaluation function warrants more exploration. A different evaluation function would yield different accuracies; as such the fitness function is high on this author's list of things to explore further.

The library functions would have required some modification to work with the binary nature of this feature selection string. It seemed more interesting to write the entire function, rather than use the library functions. This wouldn't necessarily be advisable in production code, but is interesting for the purposes of a research project.

After the mentioned modifications, accuracy for the genetic algorithm selected feature subset improved (see charts and tables below). The accuracies never reached the hoped-for levels of ~70% or better, but it was interesting to compare the accuracies using the entire feature vector and the subset selected by the genetic algorithm. The accuracies for the feature subset were more stochastic than the relatively smooth line in the results using all features.

Accuracies for the compared metrics can be seen below. Multiple runs of the program were done to find out average performance; the results here represent a typical program run.

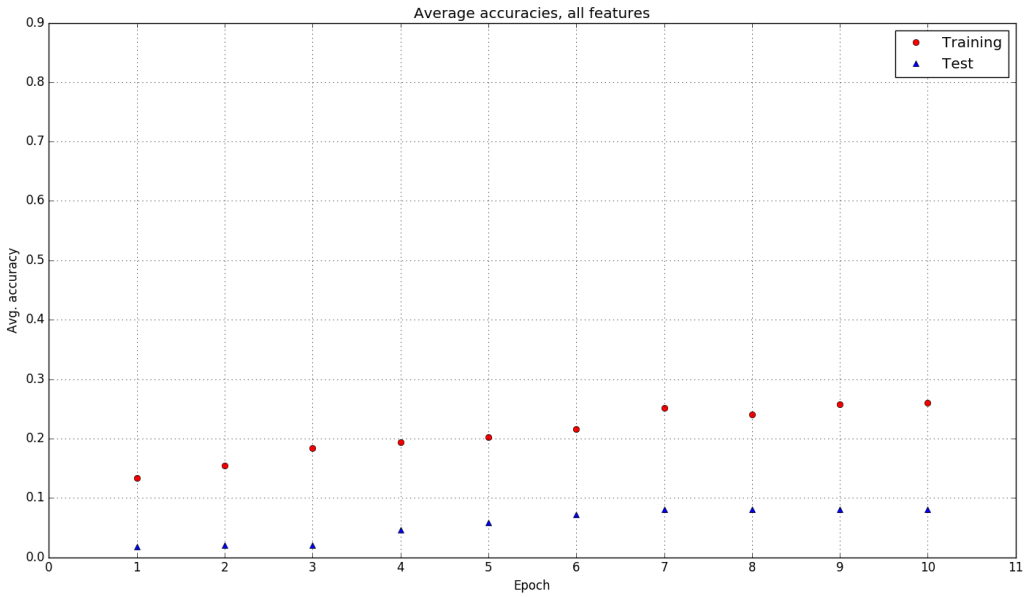

Figure 3: Accuracy using entire feature vector, using average accuracy per epoch over multiple rounds of epochs

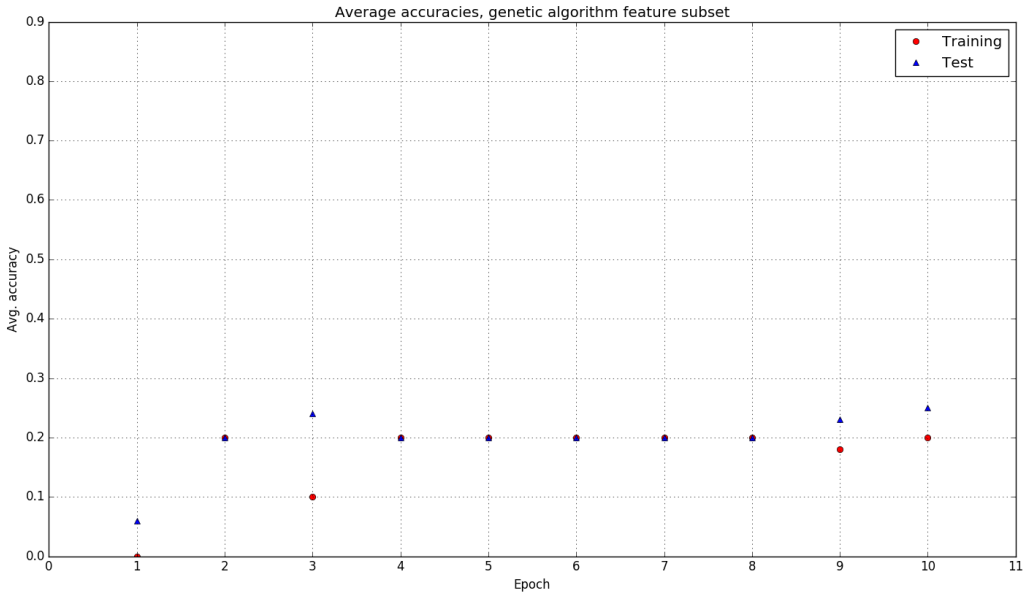

Figure 4: Accuracy using genetic algorithm selected feature subset, using average accuracy per epoch over multiple rounds of epochs

Grand Means: training accuracies	
All features	**Genetic algorithm feature subset**
0.2094	0.168

Table 1: Comparison of accuracies vs. number of features in training

Grand Means: test accuracies	
All features	**Genetic algorithm feature subset**
0.0554	0.198

Table 2: Comparison of accuracies vs. number of features in test

Summary and Open Questions

Overall, the results from this experiment yielded moderate improvements over performance using the entire feature vector. Training accuracies were approximately the same when comparing performance between the entire feature vector and a subset. There was a significant jump in test accuracies (see Table 2) when using the genetic algorithm feature subset. It is likely there is a problem in the back-propagation algorithm used to get accuracy for the entire feature vector. This would skew results.

The improved accuracy is still promising. The neural net performed faster (in the order of a few seconds) using a feature subset, and saw no significant decrease in accuracy. This indicates that the feature subset is a good way to use classifiers to recognize patterns; a result that matches the results found in Yang and Honavar [4].

Much of this project prompts continued work. Accuracies, especially using the test set, could be improved. It was hoped that there would be an improvement in speed and accuracy across classifier training. Using modified assignment code (from the class neural net assignment) was an interesting mental exercise, and was worthwhile despite the accuracy issues that likely came along with the codebase. In the future, using an existing neural net library may aid in improving classifier accuracy. The availability of neural net libraries in Python is surprisingly sparse (and they are in beta in some cases), though they do exist.

As far as metric improvement for a genetic algorithm selected feature subset, speed is somewhat improved, but the accuracies need work. There are many points of potential improvement, both in the neural net and the genetic algorithm. The top priorities of this particular codebase are back-propagation and the genetic algorithm fitness function.

The medical applications mentioned as being of practical importance have not been tested in this application; future versions could apply the neural net developed here to a different set of problems (versus the letter recognition here). This author plans to keep working on this project.

Open questions and potential further research involve topics related to pattern classification using structures like neural nets. Avenues for further research mentioned in Yang and Honavar include gene sequencing, data mining, and knowledge discovery [4, p.12]. The author of this paper has particular interest in the applications of genetic algorithms to computational models. The wide potential applications of pattern recognition mean that a genetic algorithm that optimized feature selection is valuable in a number of contexts.

References

[1] DEAP (Distributed Evolutionary Algorithms in Python) Project, "DEAP Documentation" (2016). Available at http://deap.readthedocs.org/en/master/ (Accessed 14 March 2016).

[2] Lichman, M. UCI Machine Learning Repository (2013). Irvine, CA: University of California, School of Information and Computer Science. Available at http://archive.ics.uci.edu/ml/datasets/Letter+Recognition (Accessed 14 March 2016).

[3] Mitchell, Melanie, *Complexity: A Guided Tour.* (2009). Oxford University Press, Inc., New York, NY, USA.

[4] Yang, Jihoon and Honavar, Vasant, "Feature Subset Selection Using a Genetic Algorithm" (1997). Computer Science Technical Reports. Paper 156.
Available at http://lib.dr.iastate.edu/cs_techreports/156 (Accessed 14 March 2016)

Code for this project available on GitHub:
https://github.com/abrahamsk/ml_feature-selection-ga/tree/master/neural-net-ga
(Accessed 17 March 2016)