Machine Learning Final Project

Bryce Harmsen

June 10, 2020

Introduction

In this experiment, four classifiers were compared based on run time and ability to correctly classify test samples. These four classifiers, Multilayer Perceptron, Support Vector Machine, Radial Basis Function Network, and Random Forest, were run using the WEKA library in Java. To provide each classifier a reasonable chance at properly classifying the Python-generated noisy sample data, the hyperparameters of each classifier were tuned through a Random Search optimization algorithm. Range choices for hyperparameters and conclusions about preferred classifiers based run times and classification abilities are discussed throughout the Results and Analysis sections. The data set used for this classification experiment is defined in the Implementation section.

Implementation

The machine learning algorithms and surrounding work were implemented through two programs: a sample generator written in Python and a hyperparameter optimizer that operates on a set of WEKA-based machine learning algorithms in Java.

Python

The sample generator, **samplegenerator.py**, copies a template .arff file and some file-based parameters to generate noisy samples based on 7 x 7 prototype vectors for letters A through J. The noisy samples have $7^2 \cdot max\_noise$ bits inverted. These sample sets are randomly divided into two sets of vectors, a training set and a testing set. The training sets contain 60% of the samples and the testing sets contain the other 40% of the samples generated. The training and testing sets are paired up in sub-directories within the **arffs** directory. The sample generator code also has a suite of unit tests within **test_samplegenerator.py**.

WEKA

The machine learning algorithms are all run in Java using the WEKA library. The machine learning project exists within the **maven/ml/src/main/java/ml/** internal path in the project. The high-level actions of the application exist in **App.java** and **ClassifierRunner.java**. The application runs a random search optimization on a subset of the hyperparameters for each classifier. The subset of hyperparameters are determined in the concrete subclasses of the abstract class **OptionsBuilder**. Logs of each execution of the application are saved in the **logs/** directory. All results generated by an execution of the application are saved in the **results/** directory.

While WEKA only offers one implementation of multilayer perceptron and random forest, SVM and RBF are available through two different implementations. In this implementaion, the SMO and RBF Network implementations were chosen, as opposed to the LibSVM and RBF Classifier implementations. The RBF Network was chosen because of its similarity to the network described in class. The SMO implementation was chosen because it came default with the WEKA library and did not require additional installation.

Results

Below are the best results found during 30 iterations of Random Search hyperparameter optimization. Each classifier was tuned based on a unique set of hyperparameters and for nine different configurations of sample size and percentage of noisy data. These nine configurations are combinations of three different values of each of the n_samples and max_noise parameters.

Multilayer Perceptron

The results for the Multilayer Perceptron (MLP) are listed by number of samples per prototype and maximum percentage of noise to be created in each sample. In table 1, MLP shows the best response to lower noise. Notice that the percent of correct predictions is affected negatively by an increase in percentage of noise. This is most evident in the third row, when 30% of the sample bits are inverted. The threshold for degradation based on noise appears to be between 20% and 30%. The classifier responds similarly to all available sample sizes. Times vary in the expected manner, with execution times increasing as the number of samples increases.

n samples $\rightarrow$	30	50	100
max noise $\downarrow$			
10%	16.1 s	25.02 s	43.0 s
	98.33%	100%	99%
20%	14.2 s	27.1 s	44.6 s
	91.67%	89.5%	94.25%
30%	17.7 s	31.3 s	47.9 s
	65.83%	69%	69.5%

Table 1: MLP Results (time in seconds and percent correct)

Support Vector Machine

The Support Vector Machine (SVM) results in table 2 show that the SVM classifier responds to the percentage of noise, though not consistently. The table shows that sometimes the classifier did a very poor job at correctly classifying the test results. We can see that only 10% of the tests were correctly classified. This will be explored in more detail in the Analysis section.

n samples $\rightarrow$	30	50	100
max noise $\downarrow$			
10%	2.9 s	1.6 s	1.9 s
	96.67%	99%	98.25%
20%	1.9 s	2.4 s	1.8 s
	10%	87%	10%
30%	1.9 s	2.2 s	1.7 s
	71.67%	10%	66.5%

Table 2: SVM Results (time in seconds and percent correct)

Radial Basis Function Network

The Radial Basis Function (RBF) Network responded most to the percentage of noise. Notice in table 3, the percentage of correct predictions decreases between 20% and 30% maximum noise. The RBF Network was not noticeably or consistently affected by the number of samples per prototype.

n samples $\rightarrow$	30	50	100
max noise $\downarrow$			
10%	34.6 s	34.7 s	50.8 s
	98.33%	100%	99.25%
20%	32.6 s	40.7 s	65.3 s
	92.5%	89.5%	96%
30%	32.0 s	45.7 s	74.3 s
	74.17%	75%	74.5%

Table 3: RBF Network Results (time in seconds and percent correct)

Random Forest

Similar to other classifiers, Random Forest shows the most consistent and noticeable response to the percentage of noise introduced. While all results in the 10% and 20% trials achieved over 90% accuracy, the percentage of correct predictions decreased dramatically between 20% and 30% maximum noise. Notice that the number of samples shows no pattern of effect on the percentage of correct predictions.

n samples $\rightarrow$	30	50	100
max noise $\downarrow$			
10%	0.9 s	1.3 s	2.4 s
	99.17%	100%	99.5%
20%	1.0 s	1.7 s	3.4 s
	92.5%	92%	95.75%
30%	1.1 s	1.9 s	3.6 s
	75%	71%	74%

Table 4: Random Forest Results (time in seconds and percent correct)

Analysis

Each of the classifiers assessed in the experiment provide hyperparameters that can be adjusted by the user. In an attempt to optimize the results of each of the classifiers, each classifier was run for multiple trials. Each trial executed with a different configuration of hyperparameters. While the range of each hyperparameter was chosen by the author, the actual hyperparameter values were chosen randomly within their respective ranges. This Random Search Optimization was run for 30 iterations, as mentioned above in the Results section. The optimization was run for 30 iterations to both create a viable sample size and cut down on run time. Here we will look at the randomly chosen hyperparameters that resulted in the highest percentage of correct predictions in an effort to gain insight into optimal configurations for the given problem type.

Multilayer Perceptron

Random search tuned the learning rate, training time, and hidden layer neuron configuration of the MLP. While there are other hyperparameters available for tuning, these three are essential to the structure of the MLP.

First, notice that the most successful hidden layer structures were single layers containing a moderate to large number of neurons. The range for the hidden layer architecture was 1 or 2 hidden layers, each with a range from 1 to 19 neurons. This hints that adding additional hidden layers is likely not worth exploring. Based on the number of times that the optimal architecture contained the maximum possible number of

neurons for this experiment, 19, it would be worthwhile to explore a larger range of neurons in the single hidden layer.

The training time and learning rate were set to ranges of 50 to 500 and 0.1 to 0.4, respectively.These ranges seemed to be secondary factors in the optimization process. Notice that values across both ranges were a part of optimal configurations in table 5.

n samples	max noise	learning rate	training time	nodes in hidden layer(s)	percent correct
30	10%	0.3269	203	9	98.33%
50	10%	0.3895	323	10	100%
100	10%	0.3228	112	7,15	99%
30	20%	0.1172	326	11	91.67%
50	20%	0.1696	238	19	89.5%
100	20%	0.1462	56	16	94.25%
30	30%	0.1689	274	19	65.83%
50	30%	0.3977	476	19	69%
100	30%	0.3403	337	16	69.5%

Table 5: MLP Optimal Hyperparameter Configurations & Results

Support Vector Machine

For SVM, the Random Search Optimization tuned the c value, tolerance parameter, and epsilon. The range for c was 0.1 to 10. The tolerance range was from 0.1 to 0.4. The epsilon range was from 0.1 to 10.

From table 6 we can see that values from nearly the full spectrum of the available c range found their way into different optimal configurations for the various sample sizes and percentages of noise. Similarly the tolerance parameter used the full range of possible values, though there appears to be a tendency toward the upper end of the range, with most tolerance values closer to 0.4 than they are to 0.1. Also, even though epsilon values were able to be as high as 10, we see that the majority of the optimal configurations had a much lower epsilon value, and the one high epsilon value contributed to an incapable classifier.

Notice also that more than once no reasonable candidate classifier could be found. There are three different classifiers that were the optimal during their hyperparameter random search, but still resulted in an extremely low number of correctly classified test samples. The only easily noticeable common factor among these three classifiers is the size of epsilon. These three classifiers have the highest epsilon values among all optimal classifiers in table 6. In future research, it would be worth exploring a smaller range of epsilon values, with a maximum value of 0.3 or so. There is a possibility that because there were only 30 samples for each random search, and the range of reasonable epsilons were such a small percentage of the overall epsilon range, some of the optimization rounds were never offered a reasonable epsilon value.

n samples	max noise	c	tolerance	epsilon	percent correct
30	10%	4.6308	0.1225	0.0625	96.67%
50	10%	5.0876	0.3037	0.1234	99%
100	10%	1.8309	0.3031	0.2155	98.25%
30	20%	3.3005	0.3871	0.9251	10%
50	20%	0.7714	0.2794	0.2359	87%
100	20%	5.6958	0.2469	0.4085	10%
30	30%	4.6359	0.3431	0.0168	71.67%
50	30%	8.3374	0.3943	9.9066	10%
100	30%	6.2921	0.2484	0.1560	66.5%

Table 6: SVM Optimal Hyperparameter Configurations & Results

Radial Basis Function Network

The RBF Network classifier was tuned by the minimum standard deviation and ridge parameters. Overall, results were among some of the best of the given classifiers. The ranges examined for each hyperparameter were as follows: 0.1 to 0.4 for the ridge and 0.1 to 10 for the min. std. deviation.

While the minimum standard deviations among the nine various configurations of sample size and percentage of noise were spread across the available range, the optimal standard deviations tended to be in the lower half of the available range. Especially when the noise percentage was low and the classifier was achieving higher percentages of correct classifications, the standard deviation stayed at or below roughly 3.5. This can be seen in table 7. The ridge appears to have less effect on the optimization of the classifier. The values were spread fairly evenly across the available range.

n samples	max noise	ridge	minimum std. deviation	percent correct
30	10%	0.1597	0.9084	98.33%
50	10%	0.3961	2.7834	100%
100	10%	0.3605	1.5100	99.25%
30	20%	0.1945	3.1416	92.5%
50	20%	0.3340	3.5202	89.5%
100	20%	0.2818	2.2895	96%
30	30%	0.2312	5.9010	74.17%
50	30%	0.3994	9.5167	75%
100	30%	0.3030	2.2454	74.5%

Table 7: RBF Network Optimal Hyperparameter Configurations & Results

Random Forest

The Random Forest classifier was tuned based on the number of features and the bag size percentage. The range set in this experiment for the number of features was from 1 to 10. The range for the bag size percentage was from 5% to 100%.

While bag size percentages from across the available range appeared in optimal configurations, the highest achieving configurations tended to have bag size percentages in the 20% to 40% range. When the data became more noisy, the model struggled to classify the test data and tended to train with a higher bag size percentage. The number of features tended to have less obvious influence in optimization since a wide range of numbers of features showed up in the optimal results of table 8.

n samples	max noise	bag size percentage	number of features	percent correct
30	10%	31%	8	99.17%
50	10%	28%	3	100%
100	10%	42%	5	99.5%
30	20%	34%	4	92.5%
50	20%	22%	3	92%
100	20%	22%	9	95.75
30	30%	96%	8	75%
50	30%	95%	4	71%
100	30%	73%	2	74%

Table 8: Random Forest Optimal Hyperparameter Configurations & Results

Conclusion

Based on the results found through Random Search Optimization of hyperparameters the four classifiers examined, the best performance came from Random Forest. Though RBF Network produced similar percentages for correctly classified test samples, Random Forest built classifier of similar performance 20 to 30 times faster. Unfortunately, possibly due to improper tuning ranges set by the author, SVM did not provide reliable enough classification percentages to be among the best classifiers. Based on the training times posted by the SVM, the SVM classifiers might have been the most preferred among the four examined here based on run times. Notice that the SVM run times did not increase consistently based on the number of samples. This could hint at an ability of SVM to scale better than the other three classifiers. In further research, it would be worthwhile to explore finer tuning of the SVM hyperparameters and compare larger sample size run times to those of Random Forest.