NGUYEN, Réal **COMP 472 SEC F**
ID#27566263 **DUE 18/11/2018**

MINI-PROJECT 2 REPORT

1. INTRODUCTION

The purpose of this report is to analyse the performance of three machine learning algorithms using Weka, along with the impact on their performance when experimenting with different hyper-parameters. The algorithms are applied to two different data sets, one containing 51 output classes, the other containing 10 output classes, and both sets containing 1024 features. This report will include an introduction to the algorithms and the hyper-parameters that I experimented with; an analysis of the data collected while experimenting; and what I have learned from this mini-project, along with what I would have liked to do if I had more time with this mini-project.

2. BASIC EXPERIMENTAL SETUP

In this section and the sections that follow, assume that "better performance" implies better accuracy, better precision, better recall, and better F-measure. Assume that every hyper-parameter is tested in isolation, with all the other hyper-parameters set to their default values. The only hyper-parameters that are considered in this report are the ones that can potentially positively affect the algorithms' performance.

Run times are negligible, unless noted. The actual results of changing the hyper-parameters will be discussed in the next section.

2.1 Naïve Bayes Classifier

The simplest algorithm of the three, the Naïve Bayes classifier does not have any hyper-parameters that directly affect its performance, whether negatively or positively. Thus, the model used to train the algorithm on both data sets is the default one as set by Weka; used for training, validation, and testing.

2.2 J48 Decision Tree

Before looking at the hyper-parameters used, let us look at the choice of algorithm for the decision tree (DT). Weka has a multitude of DT algorithms to choose from, and J48 proved to be the most reliable in terms of performance. Being based on the C4.5 algorithm, it is one of the most popular machine learning algorithms, being ranked #1 in popularity according to a paper by Wu et al. (2008). Some other DT algorithms, like REP Tree, did not perform as well, or had a 0% rate of correctly identifying some classes. Also, the Random Tree and Random Forest algorithms

were very obviously overfitting the training data sets, having either 99.9% or 100% accuracy every time.

J48's hyper-parameters that have positively impacted its performance are: confidence factor, minimum number of objects, and unpruned. Note also that for DT algorithms, having a shorter DT is favourable, and thus was also accounted for while experimenting with hyper-parameters.

The confidence factor (CF) hyper-parameter incurs more pruning the lower it is. In other words, the lower the CF is, the shorter the DT is. This will also slightly increase performance, up to a certain threshold.

The minimum number of objects hyper-parameter is the minimum number of instances per leaf. Like the CF, it will increase the performance of the algorithm up to a certain threshold and will consistently reduce the size of the tree the higher it is.

The unpruned hyper-parameter will make it so that the branches with insignificant probabilities in the DT are left as is. This will remove the confidence factor, incur a larger DT, and affect the performance. For the first data set, the performance is slightly better, for the second one, slightly worse.

The hyper-parameters used for the first data set are unpruned, set to true; and the minimum number of objects, changed from the default of 2 to 3. For the second data set, the confidence factor is changed to 0.01; and the minimum number of objects is set to 4.

2.3 Multilayer Perceptron

Multilayer perceptrons (MLP) are much more complex than the Naïve Bayes classifier and decision trees, but the reason why I picked this algorithm is because it is known to be quite good at handling more complex problems. Both data sets have 1024 features, the first data set has 50 classes, and the second one 10; which is decently complex. The trade-off is that it is significantly slower than both the Naïve Bayes classifier and decision trees. The latter two algorithms run in merely seconds, but MLP takes several minutes, sometimes hours, depending on its hyper-parameters.

The main hyper-parameters I have experimented with are decay, hidden layers, and training time.

Decay is a hyper-parameter that allows the learning rate to decrease over time, to prevent weights from getting too large. Although there should theoretically be a decay rate that determines how much the learning rate decreases, in Weka, this hyper-parameter is reduced to a Boolean option. Turning on decay does improve performance; however, the actual decay rate is unknown and cannot be modified.

The hidden layers hyper-parameter determines both the number of hidden layers and the number of nodes per layer. This is most likely the most crucial hyper-parameter. Having the parameter set to 0 makes the multilayer perceptron act identically to a single layer perceptron and will not perform very well. Similarly, having too few nodes on one or multiple hidden layers

will also result in bad performance. In general, the more hidden nodes, the better, but complexity and run time are exponentially increased.

Training rate is the number of epochs the MLP goes through. This constantly decreases the error rate the higher it is and increases performance up to a certain threshold. Once past that threshold, the network is said to be "over-trained" and performance decreases, meaning that the network would run for longer at a loss.

For both data sets, there is one hidden layer with the average (a) between the number of input and output nodes. This is one of the possible wildcard values defined by Weka. a = 537 for data set 1 and a = 517 for data set 2. Both data sets go through 10 epochs of training and the only adjusted hyper-parameter is decay, which is set to true.

3. ANALYSIS OF RESULTS

In this section, data set 1 and data set 2 are abbreviated DS1 and DS2, respectively.

Before diving into the details of each algorithm and their performance, it is important to note that DS1 is balanced and DS2 is unbalanced. The distribution for each class in DS1 is roughly equal, but the distribution for DS2 is skewed towards classes 1 (alpha) and 9 (xi). During validation, some algorithms cannot correctly predict some classes (2, 3, 6, 7; beta, sigma, lambda, omega, respectively) because there are so few instances of them. In other words, all predictions of these classes are false negatives, so precision and F-measure cannot be calculated.

All tables included in this section use a weighted average for all metrics rather than the regular average. Metrics are taken for each class, and the weight corresponds to the number of instances for that same class. The weighted average for every class i is: $\Sigma([\text{metric}]_i * \text{weight}_i)$ / total number of instances.

Here is a simple example, using accuracy as the metric:

Class	Accuracy	Weight	Accuracy * Weight
0	0.6	5	3
1	1	15	15
Sum		20	18

In this example data set, there are 5 instances in class 0 and 15 instances in class 1, for a total of 20 instances. If we calculate the regular average of the accuracy of the two classes, we would get: (0.6 + 1) / 2 = 0.8. However, the *weighted* average accuracy calculated by Weka would be: 18/20 = 0.9.

On all algorithms, there seems to be a sharp increase in performance when testing on DS2 vs. when testing on DS1 (refer to the tables in this section). This is because DS2 is unbalanced, and

therefore has much greater weights on the classes with more instances (classes 1 and 9). Because of this, the weighted average of the metrics is also skewed.

3.1 Naïve Bayes Classifier

As mentioned in the previous section, the hyper-parameters for the Naïve Bayes classifier stayed the same for training, validation, and testing for both data sets. While not very sophisticated, it performs surprisingly well, especially on the validation phase, compared to the other algorithms.

Naïve Bayes Classifier - DS1

	Accuracy	Avg. Precision	Avg. Recall	Avg. F-Measure
Training	0.719	0.731	0.719	0.719
Validation	0.599	0.621	0.599	0.598

Table 1

Naïve Bayes Classifier - DS2

	Accuracy	Avg. Precision	Avg. Recall	Avg. F-Measure
Training	0.810	0.816	0.81	0.812
Validation	0.801	0.809	0.801	0.803

Table 2

The fact that DS2 is unbalanced does not significantly affect the performance of the Naïve Bayes classifier, although the overrepresented classes do tend to have more false positives because they have a higher probability.

3.2 J48 Decision Tree

For the J48 decision tree, the training and validation phases were first tested using the default hyper-parameters given by Weka. The metrics collected during these phases are appended with "(default)." The phases that have their hyper-parameters adjusted are appended with "(adjusted)." Please refer to section 2.2 for details on which hyper-parameters were adjusted and why I chose them.

J48 - DS1

	Accuracy	Avg. Precision	Avg. Recall	Avg. F-Measure	Nb. of leaves	Size of tree
Training (default)	0.861	0.867	0.861	0.86	472	943
Training (adjusted)	0.783	0.789	0.783	0.781	389	777
Validation (default)	0.344	0.352	0.344	0.339	472	943
Validation (adjusted)	0.354	0.381	0.354	0.347	389	777

Table 3

J48 - DS2

	Accuracy	Avg. Precision	Avg. Recall	Avg. F-Measure	Nb. of leaves	Size of tree
Training (default)	0.945	0.945	0.945	0.945	495	989
Training (adjusted)	0.886	0.884	0.886	0.884	240	479
Validation (default)	0.782	0.784	0.782	0.782	495	989
Validation (adjusted)	0.792	0.788	0.792	0.788	240	479

Table 4

The most apparent issue with the data collected in table 3 is the sharp decrease in performance during the validation phase, compared to what is shown in table 4. This is because of the high number of output classes (51). While both DS1 and DS2 have similar tree sizes on default settings, the higher number of classes means that leaves are much more likely to falsely classify an instance.

Compare the default and adjusted algorithms on DS2: there is a decrease in performance on training and slight increase on validation when adjusted, but the main point of interest is the number of leaves and the size of the tree. Indeed, both DS1 and DS2's adjusted algorithms have smaller DTs.

3.3 Multilayer Perceptron

In this section, the data labelled "(default)" has all hyper-parameters unchanged from Weka's defaults, *except* for the training time, which is set to 10. This is because leaving the model to run for 500 epochs and 500+ nodes would take hours to run, and anything past 10 epochs seems to over-train the algorithm.

Multilayer Perceptron - DS1

	Accuracy	Avg. Precision	Avg. Recall	Avg. F-Measure
Training (default)	0.846	0.885	0.846	0.853
Training (adjusted)	0.884	0.893	0.884	0.885
Validation (default)	0.595	0.677	0.595	0.584
Validation (adjusted)	0.646	0.682	0.646	0.644

Table 5

Multilayer Perceptron - DS2

	Accuracy	Avg. Precision	Avg. Recall	Avg. F-Measure
Training (default)	0.857	?	0.857	?
Training (adjusted)	0.981	0.981	0.981	0.981
Validation (default)	0.827	?	0.827	?

Validation (adjusted)	0.921	0.921	0.921	0.921

Table 6

The most interesting data here is in table 6. The average precision and F-measures are totally absent from the default algorithms in DS2. This is because the algorithm never predicts this class – all predictions are false negatives. This is most likely because class 6 has very few instances, and therefore very little chance to get its weight adjusted. Because the class is never predicted, there are neither true positives nor false positives for class 6, so we get precision = 0 / (0 + 0), which is undefined. Because this single metric is undefined, the F-measure for this class cannot be calculated, as well as the average precision and F-measure for all classes.

One solution to fix this problem would be to increase the weights for the underrepresented classes (on their respective output nodes), but unfortunately Weka does not allow that. There is a GUI to manually build the neural network, but the model is so large that the output nodes that need to be adjusted are not visible in the window.

Fortunately, the "decay" hyper-parameter constantly decreases the learning rate for overrepresented classes, which is very useful in an unbalanced data set. This gives the underrepresented classes more of a chance to get their weights adjusted and their instances predicted.

Other than that, both MLP algorithms for the two data sets get the best performance after adjustments.

4. CONCLUSION AND FUTURE WORK

Given that we have seen in class that the Naïve Bayes classifier is used for character recognition, it is not all that surprising to see that it performs well, but I am still pleasantly surprised, because of its simplicity and speed. It also seems to do well regardless of the distribution of the data.

Because of the number of features and output classes in these data sets, I expected decision trees to not perform very well, especially for the first data set. Performance on the second data set was better since there are only 10 output classes, but I would still prefer the Naïve Bayes classifier for this application.

I expected the multilayer perceptron to be the most performant of the three, and that is the result I got. I also expected it to be slow, but I did not expect it to be *this* slow. On my first try with the default settings, I realized that it would run for a while, so I let it do its thing in the background. I returned to Weka about an hour later, and the network had only gone through 25 of its 500 epochs. Fortunately, only 10 epochs were needed to get satisfactory test results, which only takes about five minutes.

If I had more time to work on this project, I would take more time investigate how to better identify and prevent overfitting. It is a major problem in machine learning, but for this mini-project, I spent the bulk of my time experimenting with hyper-parameters just to see how they would affect performance, rather than to see how appropriate they are for the current application.

I would also have spent more time to research which are the best hyper-parameters to tweak on unbalanced data sets, if there are any. Although I now know that unbalanced data sets significantly skew the performance metrics such that they appear better than they are, I would like to know if there are ways to adjust the learning algorithms to better adapt to underrepresented classes during training without overfitting.

Although this was a nice teaser on how machine learning work looks like, I would have liked to work on much simpler data sets to get a better idea of how each hyper-parameter affects the algorithms, or on even more complex data sets on a much better computer to see how powerful machine learning can truly be.

Finally, I would like to work on different programs because of the different hyper-parameters and customization options used. I am sure that some of the issues I ran into while collecting data are limitations on Weka that can be overcome in other programs.

5. REFERENCES

https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

Official Weka PDF and YouTube MOOC tutorials

https://www.youtube.com/watch?v=4iJYJXv5yYg

Weka JavaDocs documentation

Xindong Wu; Vipin Kumar; J. Ross Quinlan; Joydeep Ghosh; Qiang Yang; Hiroshi Motoda; Geoffrey J. McLachlan; Angus Ng; Bing Liu; Philip S. Yu; Zhi-Hua Zhou; Michael Steinbach; David J. Hand; Dan Steinberg (2008) “Top 10 algorithms in data mining.” Knowledge and Information Systems vol. 14, pp 1-37.

Sam Drazin; Matt Montag “Decision Tree Analysis Using Weka.” University of Miami.

https://stats.stackexchange.com/questions/29130/difference-between-neural-net-weight-decay-and-learning-rate

https://metacademy.org/graphs/concepts/weight_decay_neural_networks