Coursework 3 Report

Preface

- **Datasets:** The details of datasets can be found in the appendix. I have used several datasets, since I want to test the results from the different number of attributes and classes. Also, I want to see how noise can influence the algorithm.
- **Experiments:** The experiments are conducted mainly by Java codes (except Visualizer). I have set up in total `50` test cases with different options on more than `4` different data sets. Therefore, there are too many results, so that they cannot be simply shown in one or two tables. All the results and data sets can be found on my GitHub: https://github.com/sunsidi/Data-Mine

1. Variation in performance with size of the training and testing sets

Both decision tree and neural network results show that the larger the training set, the better the accuracy. First of all, when the training set is large, there will be more data to feed the algorithm. Hence, the algorithm is better trained. Also, the large training set makes the testing set relatively small. There will be fewer test instances that are not included in the algorithm.

Training Set Size	Test Set Size	Accuracy
32912	1178	72.4%
29912	4178	69.7%
23863	10227	64.2%

By comparing the same experiments on the same kind of data, but different data size, it indicates that it's more likely to overfit the data on a smaller training set. Overfitting occurs when the in sample data set error is small, but out sample data set error is large. This means when the training data set is small, the algorithm is not only learning from the essential features of the data but is also learning from the noise. Thus, the algorithm can perform very well on the training set, but cannot predict future unseen data. Because the noise in the training set doesn't necessarily exist in the test set.

There are several ways to avoid overfitting the data. In decision tree, we can prune unnecessary branches and increase the minimum number of objects on each leaf. In multilayer perceptron, we can control the number of layers and neurons. A more general method is to use `K-fold cross-validation` instead of a training and test dataset. `K-fold cross-validation` run the process `k` times, and the results are averaged to produce the final model. In this way, the algorithm has been trained `k` times on `k` different data set. Thus, the algorithm will not be heavily influenced by a single data set. This explains why the all the cross-validation experiments usually have more steady results.

2. Variation in performance with change in the learning paradigm

Decision Trees

Decision tree has a greedy characteristic, it will keep developing until all the instances at the node have the same classification. Because of its greedy characteristic, decision tree is more suitable for big data. On small dataset, it will easily cause the overfitting problem. For example, when running 10-fold cross-validation on `test_3000.arff` **(7 classes, 4178 instances)** dataset, the accuracy is only about `31%` compare to `70%` on the `top_10_balanced.arff` **(7 classes, 34090 instances)**. The reason why overfitting causes very low accuracy is that it's trying to gain unnecessary information from the noise inside the training data. By fitting these noise, it causes the low performance on the test data. However, if the dataset contains no noise or very less noise, decision tree can conduct a very good accuracy on even small data set. For example, `test_happy_10%.arff` **(2 classes, 3402 instances)** dataset produced a `74%` accuracy.

Neural Network

Neural Network is also only suitable for big data set. It requires a large number of instances to train the weights. Same reason as decision tree, a small data set will easily cause overfitting. The experiment result on `test_3000.arff` **(7 classes, 4178 instances)** data set is even worse, only `28%`. Compare to decision tree, neural network requires more tuning skill and time, since there are more options that you can adjust. In addition, trying to figure out the optimal number of hidden layers and neurons is often tricky to do. In general, running neural network algorithm is much more time-consuming. Instead of simply calculating which feature is better to start next, and dividing instances at each node, neural network has to run all the data back and forth (1 Epoch) through many iterations to obtain the well-trained weights.

3. Variation in performance with varying learning parameters in Decision Trees

J48

Despite the different size of data sets and test options, the results from all datasets share the same patterns:

- **Binary Split:** This option doesn't affect the result at all in this case. Since binary split only happens when there is a nominal attribute, but in the emotional data sets, all attributes except the class are numeric.
- **Prune:** Pruning is the technique to reduce the size of decision trees by removing sections of the tree that provide little power to classify instances. Turn-off pruning will increase the accuracy because unpruning will produce a more detailed decision tree. In another word, it's overfitting the data.
- **Confidence Factor:** Confidence factor affects the degree of pruning. A smaller confidence factor will incur more pruning. Therefore, smaller confidence factor will cause lower correct accuracy. Since it removes more sections of the decision tree. Additionally, there are two interesting things worth to mention. First, even though the confidence factor's range is `(0,1)`, but any number that is greater than `0.5` will cause the problem `WARNING: confidence value for pruning too high. Error estimate not modified`. Secondly, when set confidence factor to be `0.5`, the result accuracy is very close to the unpruned accuracy.
- **Minimum Number of Objects:** This affects the minimum number of objects per leaf. When the number increases, the accuracy decreases. The minimum number of objects per leaf is actually the minimum amount of data separation per branch. When set the number to be very small, for example `1`, we are trying to give each instances it's own classification. Thus, the smaller the number is, the more likely we are overfitting the data. The minimum number of objects somehow depends on the number of classes. When the test set has only `2` classes, changing the minimum number of objects will not affect the accuracy too much. On the other hand, when the dataset has `7` classes, there is a significant drop in accuracy as the number of objects increasing. More classes mean more complexity and more noises, so in this case, overfitting is more easy to show. The significant drop in accuracy by increasing minimum number of objects suggests that it was overfitting the data in the beginning, and then starts to underfit the data.

User Classifier

User Classifier allows the user to decide where to split the data. Instead of choosing the options, users have to decide which two attributes to start with. By selecting on the plot graph, the user can decide where to split the data. In general, the user wants to start with the two attributes that have the most contrast, so the data classes are clearly separated on the graph. This makes it easy to select where to split. However, in my case, I'm using a 10-attribute dataset with `1021` instances. Among all the attributes, there is clear contrast. Eventually, I decided to use attributes `pixel 1845` and `pixel 1951`. Since data is randomly distributed, it's difficult to only include one class in a single selection. Despite the difficulty of doing classification manually, the result is `74.8%` accuracy, which is even better than the *J48* algorithm. After carefully looking at the result details, it shows that *User Classifier* cannot recognize the minority happy class.

```
=== Detailed Accuracy By Class ===

       TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
       1.000    1.000    0.748      1.000    0.856      0.000    0.635     0.814     non-happy
       0.000    0.000    0.000      0.000    0.000      0.000    0.635     0.337     happy
Avg.   0.748    0.748    0.560      0.748    0.641      0.000    0.635     0.694

=== Confusion Matrix ===

   a   b   <-- classified as
 764   0 |   a = non-happy
 257   0 |   b = happy
```

Random Forest

Random Forest constructs multiple decision trees at training time and outputting the class that is the mode of the classes for classification. I have tried this algorithm on `2` different size of training sets with `15` different settings; and the settings include adjusting **bag size, minimum number of instances per leaf, number of attributes, and depth**. In a simple conclusion, *Random Forest* algorithm produced very steady results. All the results accuracies are around `74%`, despite the various test cases. Besides the steadiness, all the results have same defects. They cannot recognize the minority class happy very well. The reason may be the algorithm choose the mode of all decision trees, instead of the best decision tree.

4. Variation in performance with varying learning parameters in Neural Networks

Multilayer Perceptron

- **Learning Rate:** Learning rate is how quickly the algorithm abandons the old beliefs for new ones. If the learning rate is too high, the algorithm can't really learn from the data that have been processed. On the other hand, if it's too low, it will take years to train the model, and need to many data to complete the model. Thus, we want something in the middle that will perform effectively and produce high-quality results. The result from `training_happy_10%.arff` and `test_happy_10%.arff` datasets shows this concept very clearly. At a low learning rate, the algorithm classifies all the data to be non-happy. Since it is stuck in the majorities, and cannot learn from the minority examples. At a high learning rate, the algorithm starts to recognize new emotions. However, this also causes many non-happy emotions to be classified as happy.

Learn rate = 0.1	**Classified as Non-happy**	**Classified as Happy**
Non-Happy	2521	0
Happy	881	0

Learn rate = 0.8	Classified as Non-happy	Classified as Happy
Non-Happy	1705	816
Happy	450	431

On the other hand, `training_happy_30%.arff` and `test_happy_30%.arff` datasets produced some odd results. The model can never classify any non-happy emotions, except when learning rate is `0.1`. The reason may be that the training set is smaller compared to `training_happy_10%.arff`. Thus, there are even fewer examples of happy. Despite the high learning rate, there are not enough instances for the algorithm to learn the minority class. Or more likely, the model is overfitting, which means it works great on the training dataset, but perform poorly on the test set.

Learn rate = 0.1	Classified as Non-happy	Classified as Happy
Non-Happy	7502	49
Happy	2578	77

Learn rate = 0.3, 0.5, 0.8	Classified as Non-happy	Classified as Happy
Non-Happy	7551	0
Happy	2655	0

- **Number of Layers:** Hidden layers are required when the data is not linearly separable. In this case, the emotion data set is not a linearly separable dataset. Because many of the emotions share the same features, for example, the emotion happy and surprise can appear at the same time on the same face. However, after several experiments, adjusting the number of hidden layers doesn't affect the accuracy significantly. The accuracy is always around `74%`.
- **Number of Hidden Neurons:** A general rule shows that the optimal size of the hidden layer is usually between the size of the input layer and the size of the output layer [1]. However, the number of hidden neurons doesn't affect the result, too. This makes me start to think that the emotion data is actually linearly separable. At least, for the `training_happy_10%.arff` data set, because it only has 2 classes.
- **Epoch:** Epoch is the number of times the algorithm sees the training examples. It's obvious that the more often the algorithm sees the training data, the higher accuracy it will be. However, this doesn't significantly increase the accuracy that much. Some of the defects of the model cannot be simply improved by passing the training data more times into it. It has to be done with other adjustments.
- **Momentum:** In neural network, the gradient descent optimization algorithm is used to find the global minima. However, there is a high chance that we are going to stuck in the local minimum. Momentum, therefore, is used to avoid this kind of situations. Also, when working with high momentum, it's important to set the learning rate low. Otherwise, it's very likely to skip the global minima with a huge step. In summary, momentum and learning rate depend on each other. The result from `training_happy_10%.arff` and `test_happy_10%.arff` shows this very well:

Learn rate = 0.8, Momentum = 0.8, Accuracy = 46.4%	Classified as Non-happy	Classified as Happy
Non-Happy	867	1654
Happy	171	710

Learn rate = 0.1, Momentum = 0.1, Accuracy = 74.1%	Classified as Non-happy	Classified as Happy
Non-Happy	2521	0
Happy	881	0

Learn rate = 0.1, Momentum = 0.9, Accuracy = 71.1%	Classified as Non-happy	Classified as Happy
Non-Happy	2244	277
Happy	705	176

Learn rate = 0.9, Momentum = 0.1, Accuracy = 61.0%	Classified as Non-happy	Classified as Happy
Non-Happy	1583	938
Happy	391	490

- **Validation Threshold:** Validation threshold is the consecutive number of errors allowed for validation testing before the network terminates. In my case, this doesn't affect the accuracy at all, because the number of consecutive errors never surpass the threshold.

5. Variation in performance according to different metrics

When evaluating an algorithm, the first thing to look at is usually the accuracy. However, in many cases, an algorithm's performance cannot be judged only by its accuracy. Consider the following Multilayer Perceptron experiments:

Accuracy = 74%	**Classified as Non-happy**	**Classified as Happy**
Non-Happy	2521	0
Happy	881	0

Accuracy = 62%	**Classified as Non-happy**	**Classified as Happy**
Non-Happy	1705	816
Happy	450	431

The first algorithm has a much higher accuracy, but it cannot recognize happy emotions at all. The second one starts to realize that there is a new class, even though it's not good at predicting the new class, which causes the low accuracy. However, in general, I prefer the second algorithm, because it has more potential and provides more human-readable information.

Besides the confusion matrix, there also many other metrics to help evaluate the results (same experiments of the previous table examples):

- Accuracy = 74%

```
=== Detailed Accuracy By Class ===

       TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
       1.000    1.000    0.741      1.000    0.851      0.000    0.634     0.822     non-happy
       0.000    0.000    0.000      0.000    0.000      0.000    0.634     0.371     happy
Avg.   0.741    0.741    0.549      0.741    0.631      0.000    0.634     0.705
```

- Accuracy = 62%

```
=== Detailed Accuracy By Class ===

       TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
       0.676    0.511    0.791      0.676    0.729      0.150    0.628     0.816     non-happy
       0.489    0.324    0.346      0.489    0.405      0.150    0.628     0.351     happy
Avg.   0.628    0.462    0.676      0.628    0.645      0.150    0.628     0.69
```

- **FP Rate**: FP rate measures how often it predicts yes when it's actually no. For example, in the first experiment, though it has a `74%` accuracy, it also has a `74%` rate on average to recognize a happy emotion as non-happy emotion. In contrast, experiment 2 only has a `46.2%` rate to make the same mistake. When considering FP rate, experiment 2 did much better than experiment 1.
- **ROC**: Let the FP rate be the x-axis, and TP rate be y-axis, then varying threshold to create dots on the graph. The curve that connects all the dots is the ROC Curve, and the area below the curve is the ROC Area. An area of 1 represents a perfect test; an area of .5 represents a worthless test. The two experiments have very close scores for ROC Area, and a `60%` rate means both are very poor tests.
- **Precision**: Precision is when it predicts yes, how often it's correct. This means non-happy is predicted as non-happy, and happy is predicted as happy. Since the first experiment cannot predict any happy emotion, it's precision is only `54.9%`.
- **Recall**: Also known as TP rate, it measures how often it predicts no when it's actually yes. In this case, this is how often the algorithm will predict happy as non-happy.
- **F-measure**: F-measure is the harmonic average of the precision and recall. F-measure reaches its best value at 1 (perfect precision and recall) and worst at 0.

In summary, though experiment 2 has lower accuracy, it does a better performance than experiment 1, since it has better scores on **FP Rate, Precision, Recall and F-measure**.

Same evaluation can be done between algorithms. Surprisingly, when compare *J48* to *Multilayer Perceptron* on the same data set with Weka's default setting, they have a very close score on each metric.

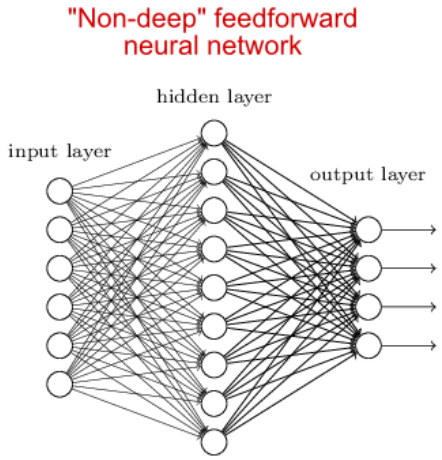
```
J48 Setting: -C 0.25 -M 2, 3402 instances
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.989    0.940    0.751      0.989    0.853      0.141    0.619     0.804     0
                 0.060    0.011    0.654      0.060    0.110      0.141    0.619     0.342     1
Weighted Avg.    0.748    0.699    0.726      0.748    0.661      0.141    0.619     0.684
---------------------------------------------------------------------------------------------
Multilayer Perceptron Setting: -L 0.3 -M 0.2 -N 500 -V 0 -S 1 -E 20 -H a, 3402 instances
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.983    0.970    0.744      0.983    0.847      0.040    0.633     0.824     0
                 0.030    0.017    0.382      0.030    0.055      0.040    0.633     0.358     1
Weighted Avg.    0.736    0.723    0.650      0.736    0.642      0.040    0.633     0.703
```

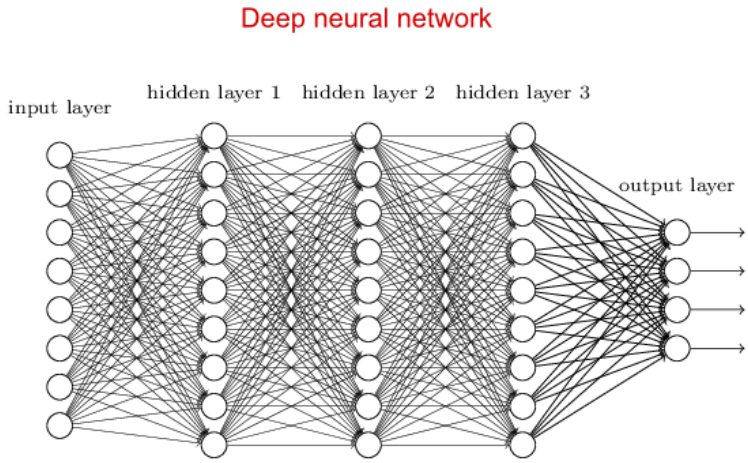
6. Comparative analysis of neural Network and Deep Neural Networks

In general, deep neural network is different from neural network because it has more hidden layers. There is not a specific definition about how many layers there has to be to qualify as deep, but usually having two or more hidden layers counts as deep [2]. On the other hand, a network with only one hidden layer is considered as "shallow".

In deep neural networks, each layer trains on the previous layer's output. Each previous layer's output can be treated as a new set of features. This feature hierarchy gives the network more capabilities to learn more complex data. This is also the reason why deep neural networks can be trained on **unsupervised** learning tasks [2]. That is to say, deep neural network is capable of discovering the intrinsic information of the data itself without a target or label. On today's internet, these unstructured and unlabeled data is the vast majority. This is why deep neural network is so popular right now.

However, does deep neural network really need that many layers to perform very well? Or in another word, can shallow network perform as good as the deeper one? The answer to this is not clear, but there is evidence shows that a shallow network with 16 layers can outperform a 150-layer network [3]. In contrast, in Ba and Caruana's paper, it states that "*with model compression we are able to train shallow nets to be as accurate as some deep models, even though we are not able to train these shallow nets to be as accurate as the deep nets when the shallow nets are trained directly on the original labeled training data*" [4]. A brave guess would be that shallow networks can perform as good as deeper one, but it's much difficult to train effectively based on the current algorithm and data information. It's worth to mention that a shallow network grows exponentially with task complexity [5]. Compare to a deep network on the same task, a shallow network will have a single layer, but with more neurons. The size of the shallow network could be even bigger than the deep network. This could be another reason that people prefer deep network to shallow network.





Appendix

1. Datasets used in the experiment

Data Set (X = test or training)	Number of Class	Number of Features	Instances	Original Data Set	Decision Tree	Neural Network	User Classifer	Random Forest
X_3000.arff	7	68	Test: 4178, Training: 29912	top_10_balanced.arff	√			
X_6000.arff	7	68	Test: 1178, Training: 32912	top_10_balanced.arff	√			
X_happy_10%.arff	2	10	Test: 3402, Training: 30618	top_10_happy.arff	√	√		√
X_happy_30%.arff	2	10	Test: 10206, Training: 20412	top_10_happy.arff	√	√		√
X_happy_noise_10%.arff	2	68	Test: 3402, Training: 30618	top10_happy_noise.arff	√	√		
X_happy_noise_30%.arff	2	68	Test: 10206, Training: 20412	top10_happy_noise.arff	√	√		
X_30%.arff	7	68	Test: 10227, Training: 23863	top_10_balanced.arff	√			
userClassifier_X.arff	2	10	Test: 1021, Training: 2381	top_10_happy.arff			√	

2. Citations

[1]: Heaton, J. (2008). Introduction to Neural Networks for Java, Second Edition. St. Louis, Mo.: Heaton research.
[2]: Introduction to Deep Neural Networks. Retrieved Nov 30, 2017, from https://deeplearning4j.org/neuralnet-overview
[3]: Zagoruyko, S., & KomodakisWide, N. (2016). Wide Residual Networks. Retrieved November 30, 2017, from https://arxiv.org/pdf/1605.07146.pdf
[4]: Ba, L. J., & Caruana, R. (2014). Do Deep Nets Really Need to be Deep? Retrieved November 30, 2017, from https://arxiv.org/pdf/1312.6184.pdf
[5]: Goodfellow, I., Bengio, Y., & Courville, A. (2017). Deep learning. The MIT Press.

3. Codes, Experiment Results and Images

My Github repository: https://github.com/sunsidi/Data-Mine